M. OURY

MAITRISEZ LE MO5

DU BASIC AU LANGAGE MACHINE

MICRO SYSTEMES

ETSF

Collection
MICRO-SYSTEMES

MAITRISEZ
le MO5

du Basic au langage machine

COLLECTION ETSF MICRO-SYSTEMES

1 - A. VILLARD et M. MIAUX, *Un microprocesseur pas à pas*
2 - A. VILLARD et M. MIAUX, *Systèmes à microprocesseur*
3 - P. GUEULLE, *Maîtrisez votre ZX 81*
4 - E. FLOEGEL, *Du Basic au Pascal*
5 - P. COURBIER, *Vous avez dit Basic ?*
6 - M. MARCHAND, *Vous avez dit micro ?*
7 - P. GUEULLE, *Pilotez votre ZX 81*
8 - M. JACQUELIN, *La micro-informatique et son A-B-C*
9 - M. OURY, *Maîtrisez les TO 7 et TO 7-70*
10 - P. GUEULLE, *Pilotez votre Oric*
11 - P. JOUVELOT et D. LE CONTE DES FLORIS, *Système d'exploitation et logiciel de base des micro-ordinateurs*
12 - P. GUEULLE, *Robotisez votre ZX 81*
13 - M. CAUT, *J'apprends le Basic*
14 - C. MALOSSE, C. TASSET, P. PRUT, *La micro, c'est pas sorcier !*
15 - R. GREGOIRE, *Bus IEEE*
16 - M. OURY, *Maîtrisez le MO5*
17 - P. GUEULLE, *Votre ordinateur et la télématique*

COLLECTION POCHE *Informatique*

1 - G. ISABEL, *50 programmes pour ZX 81*
2 - P. GUEULLE, *Montages périphériques pour ZX 81*
3 - C. GALAIS, *Passeport pour Applesoft*
4 - R. BUSCH, *Passeport pour Basic*
5 - M. ROUSSELET, *Mathématiques sur ZX 81*
6 - C. GALAIS, *Passeport pour ZX 81*
7 - G. PROBST, *50 programmes pour Casio FX-702 P et FX-801 P*
8 - G. PROBST, *60 programmes pour Casio PB-100*
9 - M. SAAL, *Utilitaires pour ZX 81*
10 - C. GALAIS, *Passeport pour Commodore 64*
11 - D. RANC, *Assembleur du TRS 80*
12 - D. LASSERAN, *30 programmes pour Commodore 64*
13 - G. ISABEL, *Du ZX 81 au Spectrum, 25 programmes*
14 - P. MELUSSON, *Initiation à la micro-informatique, le microprocesseur*
15 - G. PROBST, *40 programmes pour PB 700*
16 - C. GALAIS, *Passeport pour Basic TO 7 et TO 7-70*

Photo de couverture . Jean-Marie Aragon.

« La loi du 11 mars 1957 n'autorisant, aux termes des alinéas 2 et 3 de l'article 41, d'une part, que « les copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective » et, d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale, ou partielle, faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause, est illicite » (alinéa 1er de l'Art. 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait donc une contrefaçon sanctionnée par les Art. 425 et suivants du Code pénal ».

© 1984 – E.T S.F.

ISBN 2-85535-094-8 ISSN 0759-9498

Michel OURY

MAITRISEZ le MO5

du Basic au langage machine

Diffusion
EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 à 12, rue de Bellevue, 75940 Paris Cedex 19

Du même auteur

ETSF Micro-Systèmes

- *Maîtrisez les TO 7 et TO 7-70*

Autres ouvrages :

- *Manuel technique du TO 7 et du TO 7-70*, éd. CEDIC/Nathan.
- *Faites vos jeux en Assembleur avec le TO 7 et le TO 7-70*, éd. CEDIC/Nathan.

Sommaire

Première partie
Le Système

Introduction 9

Chapitre 1. **Le matériel** 10
Le microprocesseur 10
Les mémoires 11
Les entrées/sorties : 11

Chapitre 2. **Le logiciel** 18
Le moniteur 18
Le Basic 18

Chapitre 3. **Les extensions** 19
Le crayon optique 19
Le lecteur enregistreur de programme (LEP) 21
L'extension communication 22
Le lecteur de disquette 22
L'extension jeux 23
Le codeur modulateur SECAM 23
L'extension incrustation 23

Deuxième partie
Le Basic

Chapitre 4. **Les bases du langage** 25
Exemple introductif 26
Les différents types de variables 26
Les instructions d'entrée/sortie : INPUT, LINEINPUT, INPUT$, INKEY$, READ, DATA, RESTORE, PRINT, PRINT-TAB, PRINTUSING 27
Application 34

Chapitre 5. **La gestion de l'écran** 35
Exemple introductif 36
Les instructions de définition : ATTRB, SCREEN, COLOR, CONSOLE 37
Les instructions de positionnement : LOCATE, PSET, LINE, POS, CSRLIN, SCREEN (X, Y) 42
Application 45

Chapitre 6. **Les caractères utilisateurs** 46
Exemple introductif 49
Les instructions de définition : DEFGR$, CLEAR,,N 49
Applications 50

Chapitre 7. **Branchements, itérations et sous-programmes** 55
Exemple introductif 55
Les branchements inconditionnels : GOTO, ON GOTO 56
Les branchements conditionnels : IF... THEN... ELSE et les fonctions : AND, OR, NOT, XOR, <, >, =, EQU, IMP 58
Les itérations : FOR... TO... STEP, NEXT 59
Les branchements à des sous-programmes : GOSUB, RETURN, EXEC, CLEAR, AD 61
Applications 63

Chapitre 8. **Les variables numériques** 65
Représentation interne 65
Opérateurs arithmétiques 67
Opérateurs relationnels 67
Les fonctions arithmétiques et algébriques : ABS, CINT, COS, EXP, FIX, INT, LOG, RND, SGN, SIN, SQR, TAN 68
Applications 69

Chapitre 9. **Les chaînes de caractères** 73
Représentation interne 73
Opérateur concaténation : + ... 73
Opérateurs relationnels 73
Les instructions de chaîne : LEFT$, RIGHT$, MID$, LEN, STR$, VAL, ASC, CHR$, FRE, CLEAR 76
Police de caractères 77
Codes spéciaux 79

Chapitre 10. **Les tableaux de variables** 80
Exemple introductif 80
Dimensionnement, occupation 80
Application 82

Chapitre 11. **Les instructions de périphériques** 85
Le crayon optique : INPEN, INPUTPEN, PTRIG, TUNE 85
Les manettes de jeux : STICK, STRIG 87
La musique : BEEP, PLAY 88
Les instructions de fichier : LIST, SAVE, SAVEM, CLOSE, OPEN, LOAD, MERGE, LOADM, INPUT#, PRINT#, MOTORON, MOTOROFF, SKIPF, POS (#N) 89
Applications 93

Chapitre 12. **Les autres fonctions** 100
Ecriture et lecture en mémoire : PEEK, POKE, VARPTR 100
Gestion des erreurs : ERR, ERL, ERROR, ON ERROR GOTO, RESUME 101
Les commandes Basic : CONT, DELETE, LIST, NEW, RUN, GOTO, SAVE, LOAD, TRON, TROFF 103

Troisième partie
Dessins et graphiques

Chapitre 13. **Les zones graphiques en RAM** 105

Chapitre 14. **Les fonctions graphiques** 109
BOX, BOXF, LINE 109
PSET, POINT, COLOR, SCREEN 110

Chapitre 15. **Applications** 111

Quatrième partie
Le microprocesseur 6809

Chapitre 16. **Caractéristiques générales** 115

Chapitre 17. **Structure interne** 117
Les accumulateurs A, B, D 117
Les registres S, U, X, Y 118
Le registre compteur programme PC 118
Le registre de page DP 118
Le registre de code condition CCR 119
Notion de pile 121

Chapitre 18. **Les modes d'adressage** 123
Adressage inhérent 123
Adressage immédiat 124
Adressage étendu 126
Adressage direct 126
Adressage étendu indirect 128
Adressage relatif court 130
Adressage relatif long 132
Adressage indexé 132
Adressage indexé indirect 138
L'instruction LEA 138

Chapitre 19. **Le jeu d'instructions du 6809** 140

Cinquième partie
Le moniteur et la programmation en langage machine

Chapitre 20. **Programmation en langage machine** 146
Le mini-éditeur « LANGAGE » 146
Listing de « LANGAGE » 148

Chapitre 21. **Le moniteur du MO5** 151
Accès aux sous-programmes du moniteur 151
Utilisation des registres de la page Ø 152
Gestion de tables 152
Le sous-programme MENU 153
Les sous-programmes graphiques : PLOT, DRAW, GEPT 153
Le sous-programme GETS 160
Le sous-programme PUTCH 161
Les sous-programmes de gestion du clavier : KTST, GETC ... 167
Le sous-programme JOYST 168
Le sous-programme PRCO 169
Le sous-programme NOTE 171
Les sous-programmes LPIN et GETL 172

Annexes 1 à 10 174

Index 196

Introduction

En achetant le MO5, vous venez de faire un premier pas vers la micro-informatique. Au fur et à mesure que vous allez découvrir votre micro-ordinateur, votre appétit va s'éveiller, puis augmenter. Le but de cet ouvrage est justement de le satisfaire. Il comblera, j'espère, les appétits d'oiseaux, comme ceux des ogres.

Les premiers picoreront quelques programmes BASIC tout au long de ces pages, créeront leurs graphiques, joueront leurs partitions, en attendant...

Les seconds accéderont très rapidement au langage machine du 6809. Ils découvriront d'abord ce microprocesseur haut de gamme dans une partie que je lui ai réservée et que j'ai voulue la plus claire possible ; puis, en fin d'ouvrage, ils pourront accéder au cœur du système, le moniteur du MO5. Ils trouveront tous les points d'entrée dans les sous-programmes, les adresses importantes de la page 0 du système...

De nombreux exemples de programmes en Assembleur jalonnent cette dernière partie, ainsi qu'un mini-éditeur pour taper vos programmes en hexadécimal.

Tous les schémas et branchements internes vous ouvrent la perspective d'extensions personnelles.

Alors, bon courage, et n'hésitez pas à m'envoyer vos critiques, vos idées ou vos réalisations à ETSF.

Michel OURY

1 Le système

Chapitre 1

LE MATERIEL

1. Le microprocesseur

Le système MO5 est organisé autour d'un des microprocesseurs les plus performants actuellement sur le marché des 8 bits, puisqu'il s'agit du 6809E (voir Annexe 1 et 3e Partie).

Ce microprocesseur est un « pseudo 16 bits » dont l'organisation logicielle est symétrique et qui possède les modes d'adressage classiques, y compris l'étendu indirect et l'indexé indirect, avec la possibilité de réaliser des branchements relatifs longs.

Ce microprocesseur surpuissant de la 3e génération est entièrement compatible avec le 6800 et tous les circuits de cette famille tant sur le plan matériel que logiciel (compatibilité ascendante).

Il est donc idéal pour la réalisation de langages évolués.

Dans le MO5, il est relié à un réseau de portes logiques (« Gate Array ») qui lui fournit un signal d'horloge Q à 1 MHz en quadrature avec un signal E.

L'oscillateur d'horloge est stabilisé par un quartz 16 MHz. Le réseau de portes reçoit donc un signal 16 MHz en provenance de cet oscillateur en fonctionnement standard, ou en provenance d'un second oscillateur piloté par tension (VCO) pour permettre la synchronisation avec une image vidéo 625 lignes afin de réaliser l'incrustation de l'image écran MO5 sur l'image vidéo.

2. Les mémoires

Mémoires mortes

Le MO5 contient une mémoire morte résidente :

- La ROM Basic qui contient le Basic Microsoft est implantée de $C000 à $EFFF, soit 12 K-octets.
- La ROM Moniteur qui contient les programmes de gestion du MO5 (gestion d'écran, de clavier...) est implantée de $F000 à $FFFF, soit 4 K-octets.

Mémoires vives

Mémoires vives dynamiques du type 4416, elles occupent 48 K-octets :

- de $000 à $1FFF se trouvent les mémoires d'écran, qui occupent donc 8 K en adressage mais 16 K-octets en contenu, puisqu'à chaque adresse se trouve à la fois un octet de mémoire FORME qui indique si un point de l'écran est un point du FOND ou de la FORME, et un octet de mémoire COULEUR qui précise la couleur des points du FOND et celle des points de la FORME définis dans l'octet précédent ;
- de $2000 à $9FFF se trouvent les 32 K-octets de la RAM utilisateur.

Les 256 premiers octets, de $2000 à $20FF sont réservés au moniteur et constituent ce qu'on appelle la page zéro du moniteur.

Les 256 suivants, de $2100 à $21FF, sont utilisés par le Basic et forment alors la page zéro du Basic.

3. Les entrées/sorties

Les entrées/sorties standards sont :

- l'écran,
- le clavier,

auxquelles on ajoute rapidement le crayon optique et le lecteur/enregistreur de programme (LEP).

• **L'écran** travaille en permanence en mode graphique. On peut donc mixer graphismes et caractères. La résolution est de 64000 points (320 × 200). Les points d'une ligne horizontale sont groupés par octets (8 points). Dans chaque groupe, on ne peut définir que deux couleurs choisies parmi les seize possibles (couleurs saturées : noir, rouge, vert, jaune, bleu, magenta, cyan et blanc ; couleurs pastel : gris, rose, vert clair, jaune poussin, bleu clair, rose parme, cyan clair et orange). On peut écrire tous les caractères ASCII et définir ses propres caractères (max. 128).

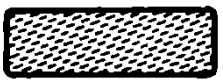
L'alphabet contient les majuscules, les minuscules, ainsi que les accents, le ç et les symboles spéciaux les plus courants. On peut doubler la hauteur et/ou la largeur de tous les caractères.

Pour la gestion de l'écran, on se reportera aux chapitres « Gestion de l'écran », « Les caractères utilisateurs » ainsi que « Dessins et graphiques ».

• **Le clavier** est du type « Goomy ». Il comporte 58 touches à enfoncement qui peuvent avoir une, deux ou trois fonctions.

1° Touches n'ayant qu'une fonction

Cette grande touche, placée au centre et en bas du clavier, est ce qu'on appelle la barre d'espace. Son rôle est de créer un espace entre deux caractères sur l'écran.



Cette touche jaune, placée en bas à gauche du clavier, est ce qu'on appelle la touche SHIFT. Elle permet l'accès aux caractères jaunes des touches à trois fonctions, par appui simultané.

Exemple :

L'appui simultané sur

et

affiche « & » sur l'écran.

L'appui simultané sur la touche jaune (SHIFT) et la barre d'espace permet l'inversion de mode MAJUSCULE/minuscule.

Cette touche valide une réponse, une commande, une ligne de programme, etc.

[BASIC]

Cette touche permet l'accès aux instructions Basic situées dans le bas des touches à deux ou trois fonctions (mot en noir). Son appui simultané avec une de ces touches permet d'écrire directement tout le mot Basic sur l'écran, sans avoir à taper les lettres l'une après l'autre.

Exemple :

L'appui simultané sur

[BASIC] et [6 & INPEN] affiche « INPEN » sur l'écran.

[CNT]

Cette touche permet l'accès aux caractères ou fonctions dont le code ASCII est inférieur (théoriquement) à 64. En pratique, elle permet principalement l'accès aux fonctions dont le code ASCII est compris entre $00 et $20.

Son action doit être simultanée à l'appui sur une lettre majuscule de l'alphabet. La fonction obtenue est celle dont le code ASCII vaut le code de la lettre moins 64 ($40).

Exemples :

CNT	C	donne accès à 67 – 64 = 03	STOP
CNT	G	donne accès à 71 – 64 = 07	SONNETTE
CNT	H	donne accès à 72 – 64 = 08	BS (←)
CNT	I	donne accès à 73 – 64 = 09	HT (→)
CNT	J	donne accès à 74 – 64 = 10	LF (↓)
CNT	K	donne accès à 75 – 64 = 11	VT (↑)
CNT	L	donne accès à 76 – 64 = 12	R à Z
CNT	M	donne accès à 77 – 64 = 13	ENTREE
CNT	Q	donne accès à 81 – 64 = 17	DC1
CNT	T	donne accès à 84 – 64 = 20	Arrêt curs.
CNT	V	donne accès à 86 – 64 = 22	ACC
CNT	W	donne accès à 87 – 64 = 23	ETB
CNT	X	donne accès à 88 – 64 = 24	CAN

On remarquera que CNTW permet le lien de deux lignes et que CNTX efface la fin d'une ligne à partir de la position courante du curseur.

2° Touches à plusieurs fonctions

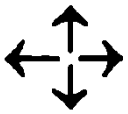
Dans ces touches, une fonction correspond toujours à une instruction ou une commande Basic, et est obtenue par action simultanée sur cette touche et sur la touche BASIC .

Les autres fonctions peuvent être des caractères, éventuellement obtenus par action simultanée sur la touche jaune, ou l'une des fonctions suivantes :

STOP (CNT-B) — Cette touche arrête le défilement de l'écran, qui reprend par une action sur n'importe quelle autre touche. Utilisée surtout pendant un listing.

RAZ (CNT-L) — Cette touche efface l'écran et ramène le curseur en haut et à gauche de l'écran.

— Cette touche ramène le curseur en haut et à gauche de l'écran, sans effacer ce dernier.

 — Ces quatre touches permettent le déplacement du curseur d'une case dans le sens indiqué par la flèche.

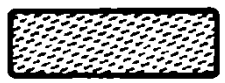
INS — Cette touche crée un espace là où se trouve le curseur. On peut alors y insérer un caractère.

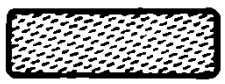
EFF — Cette touche efface le caractère courant et provoque un décalage gauche de toute la fin de la ligne.

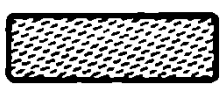
ACC — Cette touche permet le passage en mode accentué. Il existe alors deux procédures possibles :

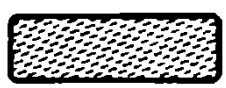
1° – taper l'accent ;
– taper la lettre à accentuer.

On obtient les accents :

– grave par action sur les touches  et

– aigu par action sur les touches  et

– circonflexe par action sur les touches  et

– tréma par action sur les touches  et

– cédille par action sur les touches et

2° – taper l'une des touches numériques ci-dessous :

[6] affiche é
[7] affiche è
[8] affiche ù
[9] affiche ç
[Ø] affiche à

• Gestion du clavier

Le clavier est géré par le PIA 6821 système que l'on peut adresser de &HA7CØ à &HA7C3, ainsi que par un multiplexeur à huit entrées, le 74LS151, et un démultiplexeur à « collecteur ouvert », le 74LS156.

Ce sont les bits PB1 à PB7 du PORTB du PIA qui gèrent entièrement le clavier selon le schéma ci-dessous (fig. 1-1).

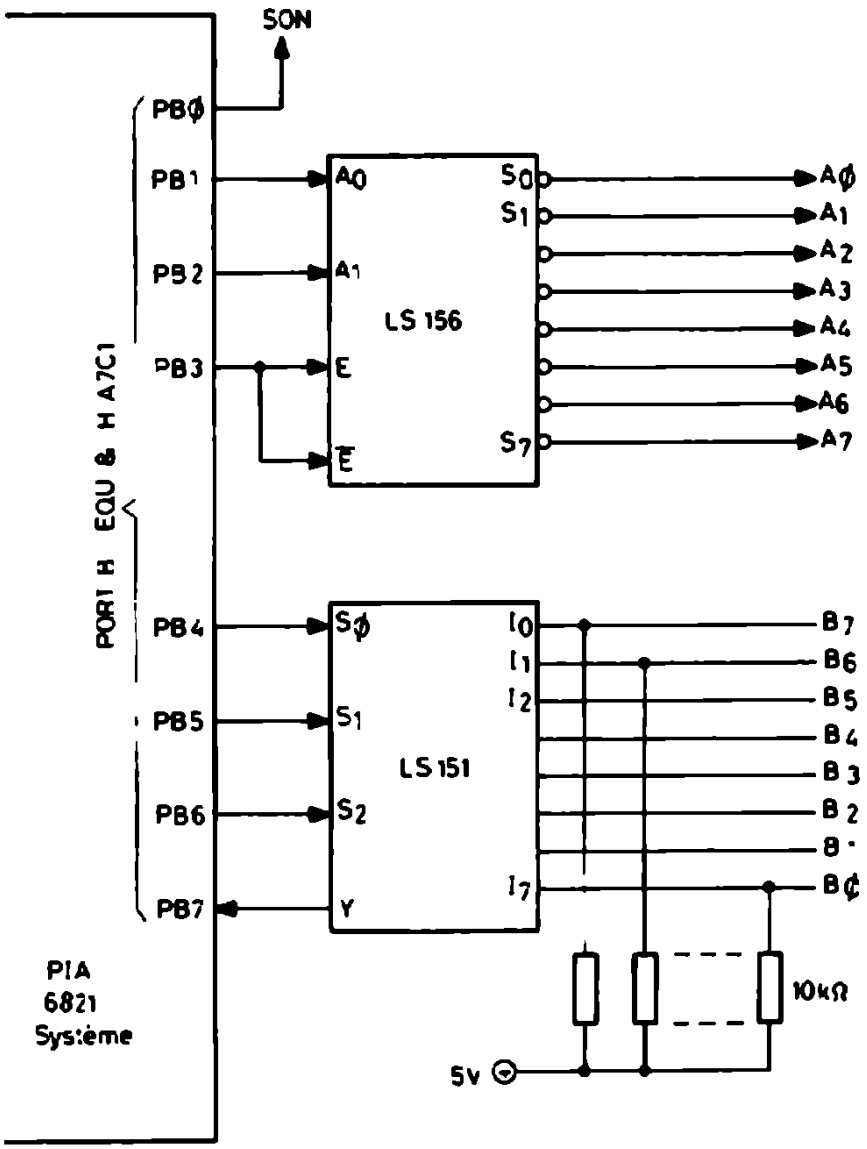


Fig. 1-1. Gestion du clavier.

Le clavier de 58 touches est organisé selon une matrice de 8 × 8, conformément à la figure 1-2.

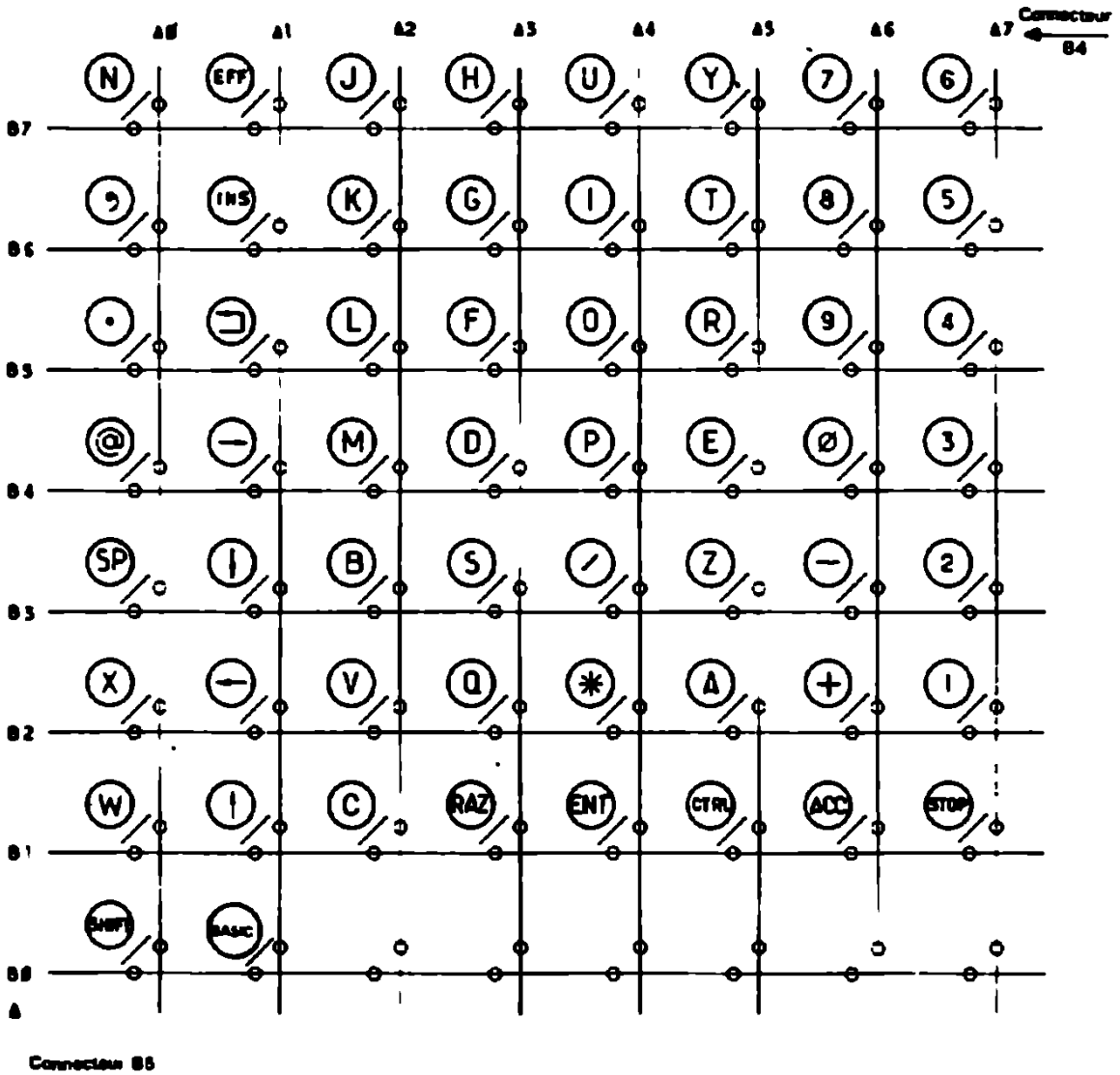


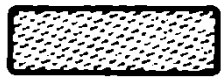
Fig. 1-2. – Organisation matricielle du clavier.

Exemple de fonctionnement

Pour faire passer la ligne AØ à Ø, il faut que PB1, PB2, PB3 soient à Ø.

Si aucune des touches reliées à cette ligne n'est enfoncée, alors toutes les lignes de BØ à B7 sont à 1 (grâce aux résistances de « pull-up » de 10 kΩ).

Dans ce cas, quel que soit le code binaire affiché avec PB4, PB5 et PB6, la valeur lue en PB7 sera 1.

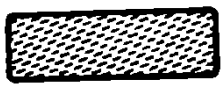
Si l'on veut détecter l'enfoncement de la touche 

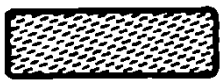
(SHIFT) qui se trouve à l'intersection des lignes AØ et BØ, il faut :

1° Mettre la ligne AØ à Ø en faisant PB1, PB2, PB3 = ØØØ.

2° Décoder la ligne BØ (c'est-à-dire l'entrée I7 du 74LS151) en faisant donc PB4, PB5, PB6 = 111.

3° Lire la valeur de PB7 :

Si PB7 = 1 → la touche  n'est pas enfoncée

Si PB7 = Ø → la touche  est enfoncée.

Chapitre 2

LE LOGICIEL

1. Le moniteur

C'est un programme en langage machine, implanté dans la mémoire morte du MO5, et dont le rôle est de gérer les échanges avec les interfaces de périphériques, puis de commander l'enregistrement et le traitement d'un programme Basic.

Ce programme contient une douzaine de sous-programmes que l'utilisateur peut appeler par EXEC en Basic, ou en utilisant une interruption « software » grâce à l'instruction SWI en langage machine.

Pour pouvoir faire fonctionner convenablement ces « routines », il est nécessaire de définir des paramètres (constantes ou variables), soit en les plaçant dans certains registres du 6809 (accumulateur B, registres d'index X et Y, etc.), soit en les plaçant en mémoire vive à des adresses connues. Dans ce dernier cas, on dira que le paramètre est placé dans un registre moniteur.

L'ensemble des registres du moniteur est en mémoire vive (RAM) de l'adresse &H2000 à l'adresse &H20FF et correspond à ce que l'on appelle la page zéro du moniteur.

On trouvera la liste et les adresses des principaux registres du système à l'annexe 4.

2. Le Basic

Dans le MO5, l'interpréteur Basic est dans une mémoire morte « résidente » de 12 Ko.

Cette mémoire occupe la plage d'adresse de &HC000 à &HEFFF.

Des cartouches de programmes peuvent être connectées au MO5. A l'introduction d'une telle cartouche, il y a désactivation du Basic et la cartouche de 16 Ko occupe alors la plage d'adresse de &HB000 à &HEFFF.

Chapitre 3

LES EXTENSIONS

1. Le crayon optique

Par l'intermédiaire d'une prise DIN 5 broches (fig. 3-1), on peut connecter un crayon optique au MO5.

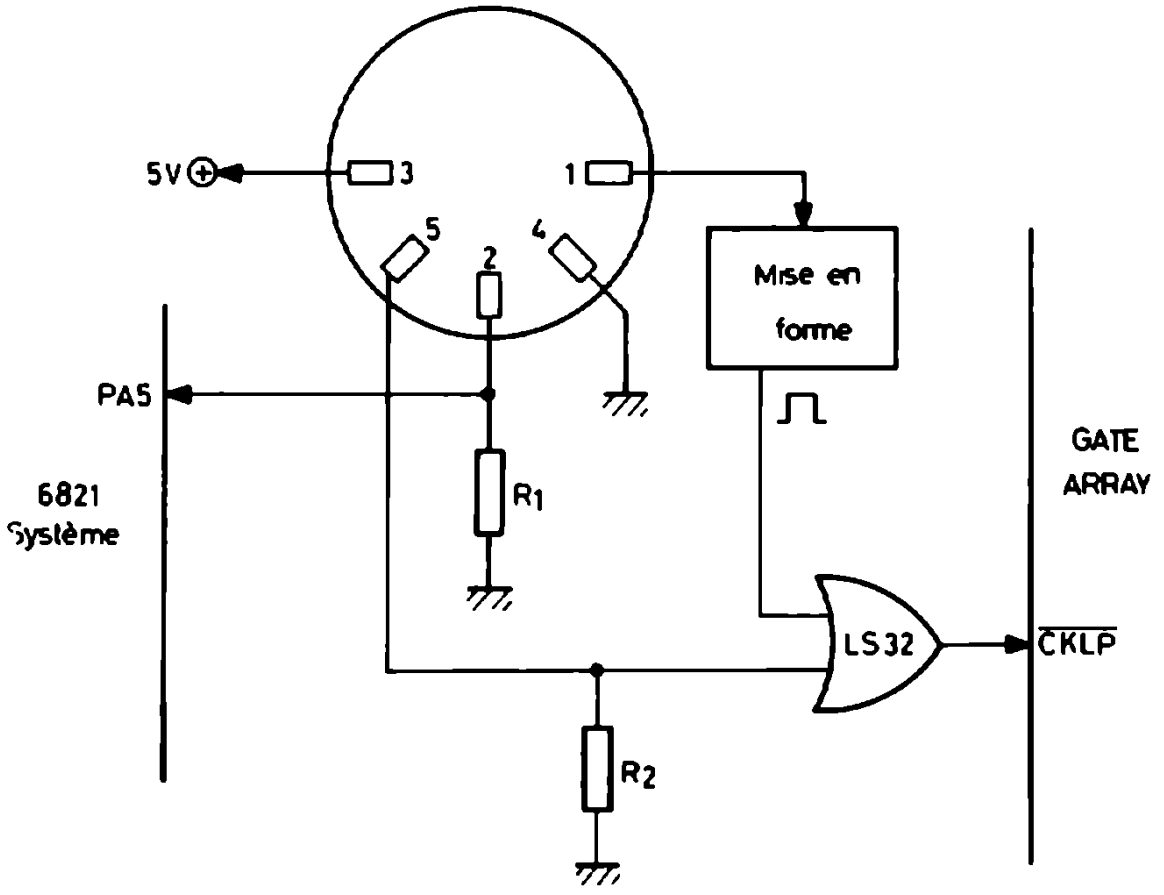


Fig. 3-1. – Brochage interne de la prise DIN « crayon optique ».

Ce crayon contient :

– Un interrupteur qui, alimenté en + 5 V par la prise DIN (broche 3), peut mettre au 1 logique, s'il est fermé, l'entrée PA5 du 6821 système (broche 2).

– Un phototransistor qui, également alimenté en + 5 V, pourra, s'il est suffisamment éclairé, demander, après mise en forme du signal, l'interruption du 68Ø9 pour lecture de la position du crayon (broche 1).

La broche 4, reliée à la masse du MO5, permet le blindage du câble de liaison.

La broche 5, inutilisée dans le crayon optique, pourra permettre une nouvelle extension à la lecture des codes barres.

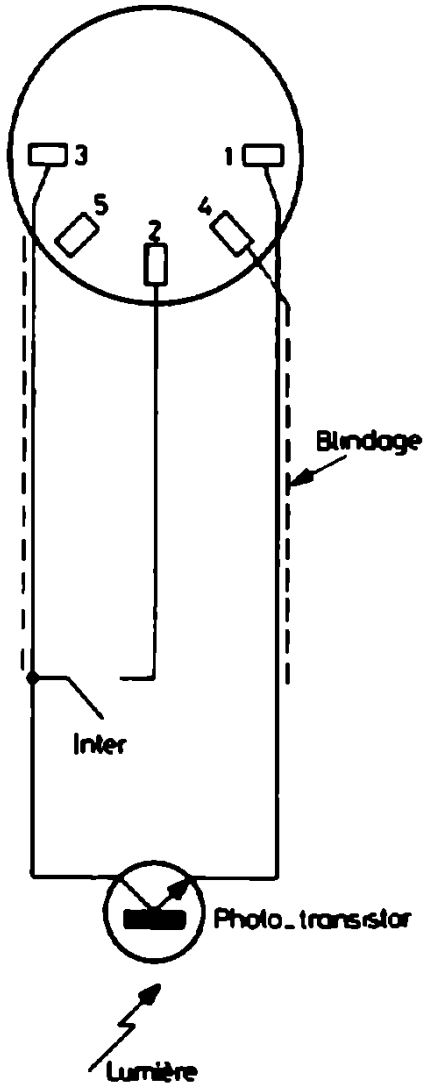


Fig. 3-2.
Brochage
du crayon optique

2. *Le lecteur/enregistreur de programmes (LEP)*

Par l'intermédiaire d'une prise DIN 5 broches, on peut connecter un magnétocassette bipiste au MO5 (fig. 3-3).

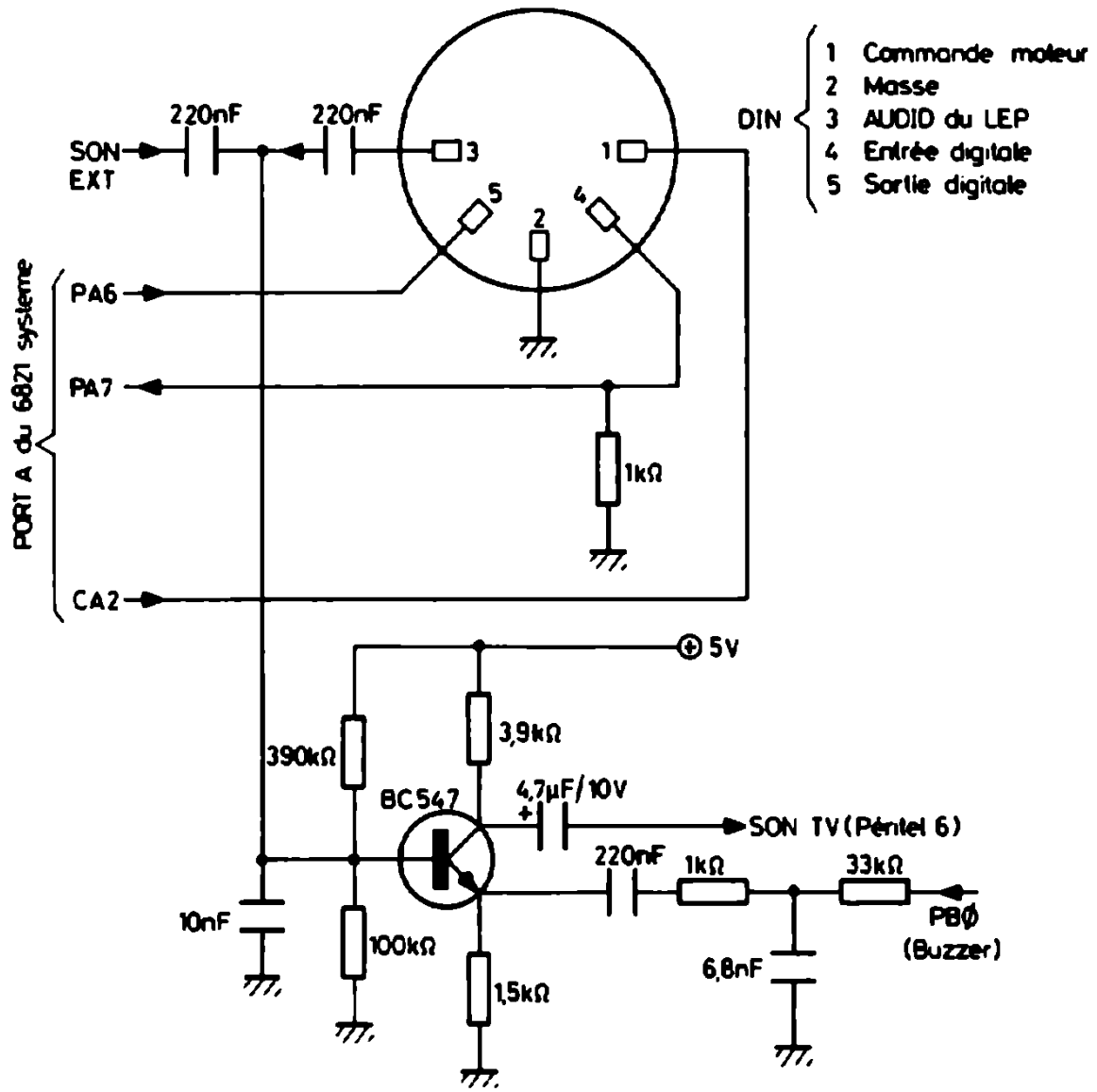


Fig. 3-3. – Brochage interne du LEP.

L'une des deux pistes sert à l'enregistrement ou à la lecture des programmes Basic, des fichiers, ou des programmes en langage machine.

L'autre piste est une piste audio, reliée via le MO5 au haut-parleur du téléviseur, et qui permet de suivre éventuellement des explications, commentaires, ou des corrections de diction pour les programmes de langue.

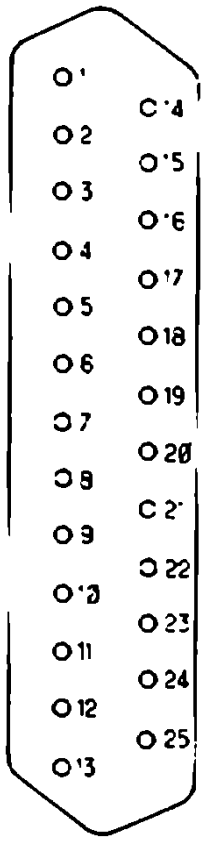


Fig. 3-4.
Prise sortie du contrôleur de communication type CANON 25.

Communication parallèle

19	BUSY
20	DATA Terminal BUSY
2	Data 1
9	Data 2
22	Data 3
10	Data 4
23	Data 5
11	Data 6
24	Data 7
12	Data 8
25	Acknowledge
1	Strobe

3. L'extension communication

Cette extension permet la liaison avec une imprimante parallèle type Centronics (fig. 3-4). Ce module extension se connecte sur le nez de carte arrière dont le brochage est donné figure 3-5.

4. Le lecteur de disquettes

Cet ensemble est constitué d'un module de contrôle qui peut gérer jusqu'à quatre lecteurs de disquettes.

Les disquettes souples de 5'' 1/4 sont du type simple face, simple densité, et comprennent 40 pistes de 16 secteurs. Chaque secteur/piste peut contenir 128 octets, ce qui conduit à une capacité de 80 K-octets par disquette.

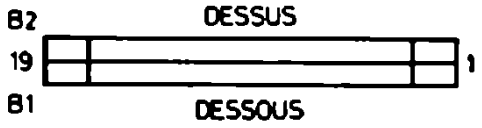


Fig. 3-5. Connecteur extensions général.

	B2	B1
1	Masse	Réservé (+ 12 V)
2	CLRG	Masse
3	A7CX	SON EXT
4	AXXX	R/W
5	A11	E
6	A10	D7
7	A9	D6
8	A8	D5
9	A7	D4
10	A6	D3
11	A5	D2
12	A4	D1
13	A3	DØ
14	A2	RESET
15	A1	FIRQ
16	AØ	VCO 16 MHz
17	BXXX	IRQ
18	SYNLT	Vidéo
19	+ 5 V	Réservé (– 5 V)

5. L'extension jeux

Cette extension comporte, outre les manettes digitales, un convertisseur digital/analogique permettant de synthétiser des sons avec quatre voix musicales indépendantes et sept octaves. Le brochage des manettes et la liaison manette PIA jeux sont donnés dans les figures 3-6 et 3-7.

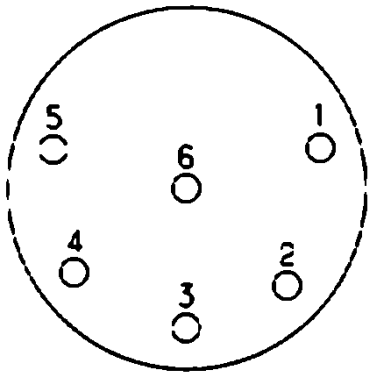
6. Le codeur modulateur SECAM

Ce codeur permet le raccordement à un téléviseur couleur via la prise d'antenne, ou à un téléviseur noir et blanc de la même façon.

7. L'extension incrustation

Ce module permet de superposer à une image vidéo analogique en provenance d'une source quelconque (émission, magnétoscope, caméra, etc) l'image du MO5 sur fond saturé noir.

Fig. 3-6.
Connecteur manettes de jeux.



1. Masse – 2. Avant
3. Arrière – 4. Gauche
5. Bouton-poussoir – 6. Droite.

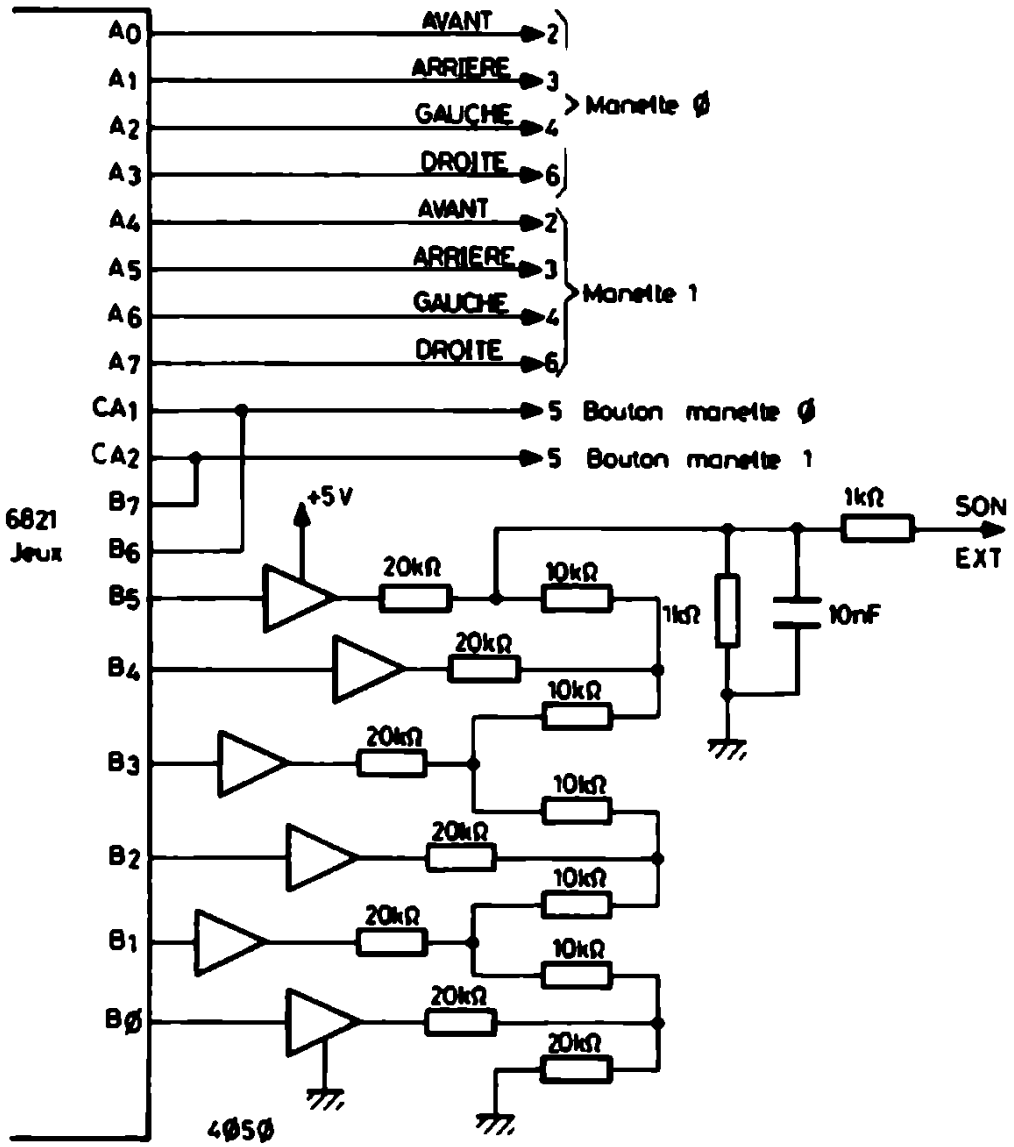


Fig. 3-7. – Liaisons internes de l'extension jeux.

2 Le Basic

Chapitre 4

LES BASES DU LANGAGE

Le langage évolué existant actuellement sur le micro-ordinateur MO5 est un Basic Microsoft 5, ce qui lui confère une très grande puissance ainsi qu'une grande souplesse d'utilisation. Ce Basic a d'ailleurs été adapté aux particularités du MO5, aussi bien sur le plan du graphisme que de la musique...

Le MO5, comme tout micro-ordinateur, est avant tout un organe de communication entre une UNITE CENTRALE et l'UTILISATEUR. Ce dernier entre des informations dans l'unité centrale, principalement à l'aide du clavier, et lit les résultats du traitement de ces informations, principalement sur l'écran.

Ce paragraphe est consacré aux différents moyens utilisables pour entrer des informations dans l'unité centrale et pour afficher les résultats sur l'écran.

1. Exemple introductif

Supposons que l'on souhaite gérer une bibliothèque contenant des romans policiers/livres de fiction/romans historiques/romans d'aventures/livres d'enfants/bandes dessinées.

L'utilisateur de cette « banque de données » doit pouvoir connaître dans chaque série : le nom de l'auteur/le titre de l'ouvrage.

Nous supposerons que la première partie du programme, celle qui présente sur l'écran le titre « BIBLIOTHEQUE » et les différents choix de types d'ouvrages numérotés de 1 à 6, existe déjà. Cette partie sera étudiée dans le chapitre de gestion de l'écran et occupera les lignes de programme de 10 à 110.

Notre programme commence donc ligne 120, et, à cet instant, l'utilisateur a tapé le numéro correspondant au type d'ouvrage choisi.

L'écran doit donc s'effacer, faire apparaître le titre de série des ouvrages choisis, puis la liste numérotée des livres en bibliothèque.

2. Les différents types de variable

2.1. Les variables numériques

– Les variables entières sont représentées par un nom de 15 caractères maximum commençant par une lettre, suivi du symbole %.

Exemple : A% ATOME1% sont des variables entières.

– Les variables réelles (ou simple précision) sont représentées par un nom de 15 caractères maximum commençant par une lettre. On peut le faire suivre du symbole !.

Exemple : LOTO RACINE! sont des variables réelles.

2.2. Les variables chaînes

Une variable chaîne est représentée par un nom de 15 caractères maximum commençant par une lettre et suivi du symbole $. La longueur maximum d'une chaîne est de 255 caractères.

Exemple : LISTE1$ est une variable chaîne.

Pour plus de renseignements, se reporter au chapitre correspondant.

ATTENTION !

Un nom de variable ne doit jamais commencer par un mot du Basic.

Exemple : TOTO est interdit car il commence par TO (voir annexe 8).

3. Les instructions d'entrée/sortie

INPUT « Chaîne » ; VAR, VAR...

Cette instruction permet à l'utilisateur de rentrer des informations dans l'unité centrale. Lorsque le moniteur rencontre cette instruction dans un programme, il écrit sur l'écran le commentaire « Chaîne » suivi d'un ? si le séparateur est le ; (si on choisit la , il n'y a pas de ? après le commentaire), puis il attend la frappe de touches.

Toutes les frappes sont recopiées sur l'écran, jusqu'à ENTREE. A cet instant, l'information est prise en compte et affectée à la première variable de la liste.

Il doit y avoir similitude entre l'information et le type de chaque variable.

On doit ensuite rentrer les valeurs de la deuxième variable, ... Il faut rentrer exactement autant d'informations que INPUT attend de variables.

Si une chaîne de caractères contient un espace en début ou en fin, une virgule..., il faut mettre cette chaîne entre deux guillemets :

Exemple :

```
10 CLS
20 PRINT"    CALCUL DE SURFACE"
30 LOCATE 0,5 : PRINT"1 - Carre"
40 PRINT"2 - Rectangle"
50 PRINT"3 - Cercle"
60 LOCATE 2,10 : INPUT"Taper 1,2, ou 3 :",C
70 ON C GOTO 100,200,300
80 '
90 '
100 '-----Surface du carre----------
```

```
110 '
120 '
130 CLS : PRINT"SURFACE DU CARRE" : LOCATE 2,10
140 INPUT"Longueur du cote :",A : LOCATE 2,14
150 PRINT"Surface :",A*A
160 END
170 '
180 '
190 '-----Surface du rectangle------
200 '
210 '
220 CLS :PRINT"SURFACE DU RECTANGLE" : LOCATE 2,10
230 INPUT"Longueur des cotes :",A,B : LOCATE 2,14
240 PRINT"Surface :",A*B
250 END
260 '
270 '
280 '-----Surface du cercle---------
290 '
300 '
310 CLS : PRINT"SURFACE DU CERCLE" : LOCATE 2,10
320 INPUT"Longueur du rayon :",R : LOCATE 2,14
330 PRINT"Surface :",3.14159*R*R
340 END
```

LINE INPUT « Chaîne » ; VAR

Cette instruction permet d'introduire dans l'unité centrale *une chaîne* de 255 caractères maximum y compris les espaces et les symboles.

On peut faire apparaître un commentaire comme pour INPUT.

Il n'y a pas de point d'interrogation.

Exemple :

```
10 CLS
20 LINEINPUT "Donnez votre reponse :",RP$
30 INPUT"Quel est votre prenom :",P$
40 LOCATE 4.10
```

```
50 PRINT P$+" a repondu "+CHR$(34)+RP$+CHR$(34)
60 END
```

INPUT$ (N)

Cette instruction permet de rentrer N caractères du clavier, y compris les caractères de contrôle, sauf CNTRL C, sans écho sur l'écran et sans être obligé de taper ENTREE .

Exemple :

```
10 CLS : PRINT"Voulez-vous continuer ? :";
20 X$=INPUT$(1)
30 IF X$="O" GOTO 40 ELSE END
40 PRINT"Bien continuons"
```

Le programme ci-dessus permet de tester si la réponse est OUI ou NON simplement sur la première lettre, soit O soit N.

INKEY$

Cette instruction permet la saisie d'un caractère au clavier, mais, à la différence de INPUT, elle n'attend pas qu'un caractère soit tapé, et n'envoie pas d'écho sur l'écran. Il est donc nécessaire de provoquer un test et une boucle pour savoir si une touche a été enfoncée.

Exemple :

```
10 CLS : PRINT"Taper un nombre entre 1 et 9 : ";
20 A$="" :                'Chaine vide
30 A$=INKEY$ : IF A$="" GOTO 30
40 X% = VAL(A$)
50 PRINT X%
60 END
```

- Ligne 20 : on définit une chaîne vide A$.
- Ligne 30 : on affecte à A$ la touche enfoncée et on teste : si A$ est toujours vide on retourne en 30.
- Ligne 40 : X$ prend la valeur numérique entière du caractère A$.

Toutes les instructions du type INPUT, ainsi que INKEY$, attendent une réponse de l'utilisateur.

Nous allons étudier maintenant un ensemble d'instructions qui permettent au programmeur de faire lire par l'unité centrale des informations contenues dans le programme.

READ VAR, VAR...

Cette instruction permet d'affecter à la première variable la première information placée dans le programme derrière l'instruction DATA, puis d'affecter à la deuxième variable la deuxième information, etc.

Les variables sont de types quelconques, mais il doit y avoir correspondance entre le type d'une variable et son information.

DATA INF, INF,...

Cette instruction placée n'importe où dans le programme signale que ce qui la suit est l'ensemble des informations qui seront lues à la suite d'un READ.

Exemple :

```
10 CLS : PRINT"Voulez-vous connaitre les horaires
?"
20 X$ = INPUT$(1)
30 IF X$="N" THEN END
40 READ D$,H$,D1$,H1$,D2$,H2$,D3$,H3$
50 PRINT D$,H$
60 PRINT D1$,H1$
70 PRINT D2$,H2$
80 PRINT D3$,H3$
90 END
100 DATA "Reseau Est","10 Octobre 1984",NANCY,20H5
3,STRASBOURG,20H55,BESANCON,20H5?
```

On remarque qu'une chaîne contenant des blancs doit être placée entre guillemets.

RESTORE N

Cette instruction pointe le READ qui suivra vers les DATA situées ligne N.

Si N est omis, RESTORE pointe vers le premier DATA rencontré dans le programme pour, éventuellement, permettre une nouvelle lecture des informations.

Exemple :

```
50 PRINT"Voulez-vous connaitre le mot de passe ? :
";
60 X$ = INPUT$(1)
70 IF X$ = "N" GOTO 100
80 READ MOT$,M
90 PRINT MOT$,M
100 DATA TROMBONE,74
110 '......
120 '.....
200 RESTORE
210 PRINT"Voulez-vous connaitre le mot de passe ?
:";
220 X$ = INPUT$(1)
230 IF X$ = "N" GOTO 250
240 READ MOT$,M
250 '...
```

Exemple :

```
10 CLS : PRINT"Voulez-vous connaitre les multiples
 de 2,3 ou 5 ? :";
20 DATA "Multiples de 2 :",2,4,6,8,10,12,14,16
30 DATA "Multiples de 3 :",3,6,9,12,15,18,21,24
40 X$=INPUT$(1)
50 DATA "Multiples de 5 :",5,10,15,20,25,30,35,40
60 X=VAL(X$)
70 IF X=2 THEN RESTORE 20 ELSE IF X=3 THEN RESTORE
 30 ELSE RESTORE 50
```

```
80 READ T$,M1,M2,M3,M4,M5,M6,M7,M8
90 PRINT T$
100 PRINT M1,M2,M3,M4,M5,M6,M7,M8
110 INPUTWAIT10;5,Z
```

Comme on a pu le constater dans les exemples précédents, l'instruction PRINT permet l'affichage d'un texte sur l'écran. Observons donc cette instruction d'un peu plus près.

PRINT TEXTE ; TEXTE

Cette instruction permet l'affichage sur l'écran des expressions qui la suivent.

Si les expressions sont séparées par un point-virgule, elles sont écrites l'une à la suite de l'autre sans espace supplémentaire.

Attention : n'oubliez pas que toutes les valeurs numériques se terminent par un espace, et commencent par un espace lorsqu'elles sont positives (sinon par –).

Chaque ligne est divisée en trois zones de 13 caractères :

Zone 1, de Ø à 12
Zone 2, de 13 à 25
Zone 3, de 26 à 38

Si les expressions sont séparées par une virgule, elles s'inscrivent sur l'écran à partir de chaque début de zone.

Si la dernière expression n'est suivie ni de la virgule ni du point-virgule, le PRINT suivant provoquera un retour à la ligne suivante colonne Ø, sinon on reste sur la ligne en cours.

Si une expression chaîne a plus de 4Ø caractères, le saut à la ligne est automatique.

Attention : une valeur numérique ne peut s'inscrire sur deux lignes.

SPC(N)

Cette instruction placée derrière un PRINT introduit *N espaces.*

TAB(N)

Cette instruction placée derrière un PRINT ou un séparateur virgule ou point-virgule positionne le curseur dans la colonne de numéro N.

Si, lorsqu'il rencontre cette instruction, le curseur se trouve dans une colonne de numéro supérieur à N, alors il effectue un saut à la ligne avant de tabuler colonne N.

Exemple :

```
10 PRINT"BONJOUR",TAB(10);"MADAME"

BONJOUR
          MADAME
```

Ecrira :	BONJOUR		← ligne L
	↑	MADAME	← ligne L+1
		↑	
	Col Ø	Col 1Ø	

En effet, BONJOUR s'affiche colonne Ø. Puis la virgule positionne le curseur colonne 13, donc TAB (1Ø) se fait à la ligne suivante.

PRINT USING IMA ; VAR

Cette instruction permet l'affichage de la variable VAR suivant une IMAGE définie préalablement.

– *IMAGE d'une valeur numérique :* A$ = « # # # # »

Toute variable numérique utilisant l'image A$ sera arrondie à l'entier le plus proche, puis affichée avec cadrage à droite.

Si la variable est plus courte que l'image, on complète avec des blancs à gauche.

Si la variable est plus longue, elle est écrite dans sa totalité et précédée du symbole %.

On peut insérer dans l'image un point décimal pour séparer partie entière et partie décimale.

Exemple : A$ = « # # # · # # ».

Les règles précédentes restent valables. On peut faire apparaître le signe en début de zone en faisant apparaître, au début de l'image, +.

Exemple :

La valeur 12Ø.18 s'écrira 12Ø.2 si A$ = « # # # · # »
ou + 12Ø.2 si A$ = « + # # # · # »

Si une variable est plus courte que son image, on peut la compléter avec le symbole * en faisant précéder l'image de (*). (Ex. : A$ = « * # # # · # # ».)

Exemple :

La valeur 12Ø.18 s'écrira
* * 12Ø.18 si A$ = « * # # # # # · # # »

L'affichage peut se faire avec la forme exponentielle si l'image est terminée par une flèche d'exponentiation.

4. Application

On résoudra le problème posé dans l'exemple d'introduction grâce au programme ci-dessous :

Exemples :

```
10 CLS : A$=""
20 A$=INKEY$ : IF A$="" GOTO 20
30 A=VAL(A$)
40 IF A=1 THEN RESTORE 110 ELSE IF A=2 THEN RESTORE 120
ELSE IF A=3 THEN RESTORE 130
50 READ NL
60 FOR N = 1 TO NL
70      READ NOM$,TIT$
80      COLOR1 : PRINT NOM$, : COLOR 2 : PRINT TIT$
90      NEXT N
100 GOTO 20
110 DATA 2,BROWN,"MISS TRANSE",DEMOUZON,BUNGALOW
120 DATA 1,ANDREVON,NEUTRON
130 DATA 1,OURY,"MAITRISEZ LE MO5"
```

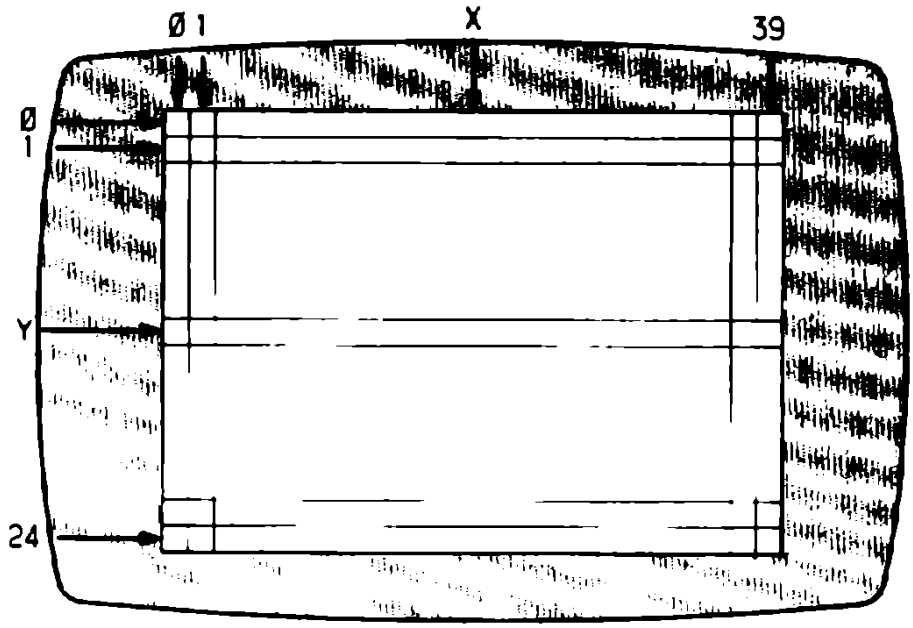
Chapitre 5

LA GESTION DE L'ECRAN « CARACTERES »

En mode caractère, l'écran peut être défini comme un ensemble de 1 000 cases réparties en 25 lignes numérotées de Ø à 24 et de 40 colonnes numérotées de Ø à 39.

Une case sera donc définie par deux numéros :

- premièrement, son numéro de colonne X
- deuxièmement, son numéro de ligne Y



ECRAN

Par exemple, la case (1), la plus en haut et à gauche, est la case Ø,Ø. Celle qui est en bas, à droite, est la case 39,24.

On remarquera qu'en fait la case Ø,Ø n'est pas en haut et à gauche de l'écran, mais en haut et à gauche de la zone écran réservée à l'ordinateur.

Autour de cette zone réservée se trouve ce qu'on appellera *le CADRE (ou TOUR),* dans lequel on ne peut en aucun cas écrire ou faire un dessin. Le cadre nous est inaccessible, mais on peut en choisir la couleur.

1. Exemple introductif

Gérer l'écran, c'est, en fait, d'abord définir :

- la couleur du cadre,
- la couleur de fond d'une case,
- la couleur d'un caractère dans une case,
- la hauteur et la largeur des caractères,
- et, éventuellement, se réserver une zone d'écriture.

Puis, il faut positionner un caractère dans l'écran pour l'écrire ou, au contraire, lire un caractère dans l'écran pour relever sa position.

Prenons l'exemple du problème suivant :

1. Sur un écran totalement noir, je veux faire apparaître simultanément :

– un titre « BIBLIOTHEQUE » en caractères rouges de double largeur et double hauteur, en haut et au milieu de l'écran ;

– des chapitres numérotés de 1 à 6. Le numéro, noir, devra toujours être dans la deuxième colonne et apparaître sur fond jaune. Les sous-titres correspondants seront en jaune sur fond noir, de taille normale et commenceront à la sixième colonne ;

– le curseur sera localisé à la case Ø,Ø.

2. L'utilisateur pourra déplacer le curseur à l'aide des touches spécialisées et l'amener ainsi devant le chapitre qui l'intéresse, n'importe où sur la ligne correspondante.

– Une action sur ENTREE devra alors entraîner la localisation du curseur, pour en déduire le chapitre choisi.

---

(1) Chaque case ne peut contenir qu'un et un seul caractère semi-graphique, lettre, chiffre, symbole, etc.

Le problème est posé ; examinons les outils dont nous disposons.

2. *Les instructions de définition*

ATTRB L, H

Cette instruction définit les attributs des caractères :

L = Ø ⇒ largeur standard
L = 1 ⇒ double largeur

H = Ø ⇒ hauteur standard
H = 1 ⇒ double hauteur

Exemples :

```
10 ATTRB 0,0
20 PRINT"Bibliotheque"

10 ATTRB 1,0
20 PRINT"Bibliotheque"
```

```
10 ATTRB 0,1
20 PRINT"Bibliotheque"

10 ATTRB 1,1
20 PRINT"Bibliotheque"
```

Après chacun de ces exemples, le lancement par RUN fait apparaître le mot dans le caractère choisi.

CLS

Cette instruction efface la fenêtre définie par l'instruction CONSOLE et ramène le curseur en haut et à gauche de cette fenêtre.

Si l'instruction CONSOLE n'a pas été utilisée, CLS efface l'écran et renvoie le curseur à la case Ø,Ø.

CLS est équivalent à l'appui sur la touche RAZ .

COLOR C, F, V

Cette instruction fixe la couleur :

C = du caractère (de Ø à 15).
F = du fond de la case où est inscrit ce caractère (de Ø à 15).

Le chiffre V (Ø ou 1) permet d'inverser la vidéo : V = Ø vidéo normale, V = 1 vidéo inversée.

Exemple :

```
10 COLOR 4,6,0
20 PRINT"A"
```

↓

Ecrira → A en bleu foncé sur fond bleu clair.

```
10 COLOR 4,6,1
20 PRINT"A"
```

↓

A en bleu clair sur fond bleu foncé.

Cette instruction n'affecte que la couleur d'une case de l'écran sans modifier les couleurs du reste de l'écran.

Code des couleurs

Les couleurs sont codées de la façon suivante :

0	1	2	3	4	5	6	7
NOIR	ROUGE	VERT	JAUNE	BLEU	MAGENTA	CYAN	BLANC
8	9	10	11	12	13	14	15
GRIS	ROSE	VERT CLAIR	JAUNE POUSSIN	BLEU CLAIR	ROSE PARME	CYAN CLAIR	ORANGE

CONSOLE H, B, C, V

Cette instruction permet de redéfinir la fenêtre écran dans laquelle on souhaite travailler, ainsi que la vitesse de « scrolling » et son type (couleur ou non).

- H représente le numéro de la ligne haute de la fenêtre.
- B représente le numéro de la ligne basse de la fenêtre.

Ainsi l'instruction CONSOLE Ø,24 nous permet de travailler sur tout l'écran, alors que CONSOLE 1Ø,2Ø, ne nous donne que 11 lignes accessibles de la ligne 1Ø à la ligne 2Ø incluses.

Dans ce dernier cas, l'instruction CLS n'efface que la fenêtre de 11 lignes ainsi définie et ramène le curseur au début de la ligne n° 1Ø.

Exemples : comparer ces deux programmes

```
10 CLS
20 CONSOLE 0,24
30 PRINT"A"
40 GOTO 30
```

```
10 CLS
20 CONSOLE 10,20
30 PRINT"A"
40 GOTO 30
```

– C définit le type de couleurs
  C = Ø ⇒ utilisation des couleurs courantes,
  C = 1 ⇒ verrouillage des couleurs.

Exemple : comparer ces deux programmes.

```
10 CLS
20 SCREEN 4,6,6
30 COLOR 1,4
40 CONSOLE ,,0
50 FOR N=1 TO 1000
60      PRINT"A";
70      NEXT
```

Dans ce premier programme, ligne 2Ø, on définit l'écran (cadre et fond bleu ciel, caractères bleus foncés).

Puis, ligne 3Ø, on définit un affichage de caractères rouges sur fond bleu ciel.

Ligne 4Ø, la console ouvre par défaut la fenêtre maximum, c'est-à-dire de la ligne Ø à la ligne 24, avec scrolling en couleurs.

La ligne 5Ø remplit l'écran de A rouges sur fond bleu foncé. Le cadre reste bleu ciel.

Si l'on modifie la ligne 4Ø de la façon suivante :

```
10 CLS
20 SCREEN 4,6,6
30 COLOR 1,4
40 CONSOLE ,,1
50 FOR N=1 TO 1000
```

```
60      PRINT"A";
70      NEXT
```

Cela annule l'effet de l'instruction COLOR de la ligne 30. Mais cette instruction a été lue et enregistrée (mémorisée) par l'ordinateur, et, si après avoir commandé (en mode direct) RUN, vous commandez CONSOLE, 0, le MO5 vous répond OK en caractères rouges sur fond bleu.

Si vous faites le programme suivant :

```
10 SCREEN 4,6,6
20 COLOR 1,4
30 CONSOLE 10,20,0
40 CLS
50 FOR N=1 TO 1000
60      PRINT"A";
70      NEXT
```

Vous ne remplissez de A rouges sur fond bleu ciel que la fenêtre 10,20.

Si vous modifiez la ligne 30 comme ci-dessous :

```
10 SCREEN 4,6,6
20 COLOR 1,4
30 CONSOLE ,,1
40 CLS
50 FOR N=1 TO 1000
60      PRINT"A";
70      NEXT
```

Vous ne modifiez pas la fenêtre qui est inchangée par défaut des constantes H et B, mais vous écrivez suivant la définition d'écran.

- V définit la vitesse du défilement (scrolling)
  V = 0 vitesse normale
  V = 1 vitesse lente
  V = 2 vitesse mode page.

Faites les trois essais dans le programme précédent en ajoutant successivement les trois valeurs de V.

```
30 CONSOLE ,,1,0
```

puis

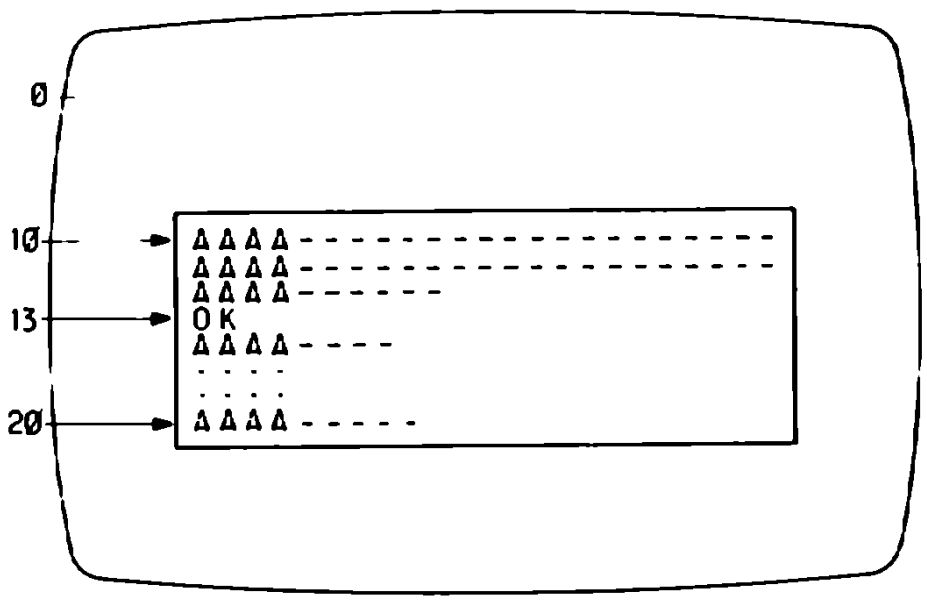
```
30 CONSOLE ,,1,1
```

et enfin

```
30 CONSOLE ,,1,2
```

Dans ce dernier cas, lorsque la fenêtre est remplie, le curseur remonte à la case départ, c'est-à-dire ligne Ø colonne Ø, et continue son impression des A. Mais comme la fenêtre ligne Ø, ligne 9 n'est pas ouverte, rien n'apparaît sur l'écran.

Or le programme est écrit pour remplir l'écran de 1ØØØ « A », soit 24 lignes. Seules 11 lignes sont remplies quand le curseur remonte case Ø,Ø. Il en reste donc encore 13, dont 1Ø (lignes Ø à 9) restent invisibles, les trois dernières réapparaissent dans la fenêtre. Puis le MO5 précise OK qui apparaît donc ligne 13.



SCREEN C, F, K, V, I

Définit la vidéo écran :

- C définit la couleur des caractères (de Ø à 15)
- F définit la couleur du fond de la fenêtre (de Ø à 15)
- K définit la couleur du cadre (de Ø à 15)
- V définit la vidéo normale Ø ou inverse 1
- I permet le mode incrusté si égal à 1.

Voir les exemples précédents.

3. Les instructions de positionnement

LOCATE C, L, Q

Cette instruction place le curseur colonne C, ligne L.

Q indique si le curseur doit être visible (Q = 1) ou non (Q = Ø).

Par défaut le curseur est visible.

Attention : les valeurs de C et de L doivent toujours être spécifiées. Elles ne peuvent être prises par défaut.

Comparez les deux programmes ci-dessous :

```
10 CLS
20 SCREEN 4,6,6,1
30 LOCATE 11,11
40 ATTRB 1,1
50 PRINT"BONJOUR"
60 LOCATE 0,22
70 ATTRB 0,0
```

↓

Dans ce cas, le curseur est visible sous l'Inscription OK.

```
10 CLS
20 SCREEN 4,6,6,1
30 LOCATE 11,11
40 ATTRB 1,1
50 PRINT"BONJOUR"
60 LOCATE 0,22,0
70 ATTRB 0,0
```

↓

Puis remplaçons la ligne 60 : Dans ce cas, le curseur est invisible sous l'inscription OK.

PSET (C, L) « K », C, F, V

Cette instruction écrit colonne C, ligne L, le caractère « K », défini par les couleurs C, F, V, identiques aux paramètres de **COLOR.**

Exemple : PSET (20,10) « $ »,1 écrit à la 20e colonne et 10e ligne un $ rouge

A$ = « 0 »

PSET (20,10) A$,0 remplace le $ par un 0 noir.

Le caractère « K » peut également être un GR$.

LINE (C_1, L_1) – (C_2, L_2) « K », C, F, V

Cette instruction trace une ligne du caractère K, de la case située colonne C_1, ligne L_1 à la case située colonne C_2, ligne L_2, dans les couleurs C, F, V.

Comme pour PSET, le caractère peut être un A$ ou GR$ ou CHR$.

Exemple :

```
10 CLS
20 SCREEN 7,0,0
30 LINE (0,0)-(38,0)"-
```

Ce programme trace une ligne pointillée en haut de l'écran en blanc sur fond noir.

POS

Cette instruction retourne le numéro de colonne du curseur.

Exemple :

```
10 FOR L=0 TO 24
20      PSET (C,L)"-"
30      NEXT
40 C=POS
50 GOTO 10
```

Lorsque vous faites RUN, toutes les variables sont réinitialisées donc C = L = 0.

De la ligne 10 à 30, on remplit la colonne C = 0 de « – ». Une fois le dernier « – » écrit sur l'écran, le curseur se positionne à la case suivante, c'est-à-dire *colonne 1*, ligne 24.

Donc, en ligne 40, C prend la *valeur 1*, etc.

L'écran se remplit donc de « – » colonne après colonne et de haut en bas.

CSRLIN

Cette instruction retourne le numéro de la ligne du curseur. N'oubliez pas qu'il y a 25 lignes numérotées de 0 à 24. Avec l'instruction précédente, on peut connaître la position exacte du curseur sur l'écran.

On peut donc reprendre l'exemple précédent en remplissant l'écran ligne par ligne.

Exemple :

```
10 FOR C=0 TO 39
20      PSET (C,L)"?",1
30      NEXT
40 L=CSRLIN
50 GOTO 10
```

SCREEN (C,L)

Cette instruction retourne le code ASCII, de 32 à 148 (voir annexe 7), du caractère situé colonne C, ligne L. Si aucun caractère connu ne se trouve en C,L, alors l'instruction retourne 0 (c'est le cas des caractères GR$ définis par l'utilisateur).

Exemple :

```
10 CLS
20 SCREEN 0,7,7
30 PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
40 FOR C=0 TO 25
50      X=SCREEN(C,0)
60      PRINTX
70      NEXT
80 END
```

4. *Application*

Le problème posé en début de chapitre sera traité dans l'ordre logique suivant :

1. Effacer l'écran/définir l'écran caractères rouge sur fond noir, cadre noir/caractère double largeur, double hauteur.
2. Se placer ligne 1, colonne 13 et écrire « BIBLIOTHEQUE ».
3. Placer le numéro 1 (noir/fond jaune), 4e ligne, 2e colonne, puis écrire « – Romans Policiers » (jaune/fond noir).
4. Sauter 3 lignes et refaire 3 avec le n° 2 et « Série Fiction », etc.
5. Placer le curseur en 0,0 et attendre la frappe sur ENTREE.
6. Lire la position du curseur.
7. En déduire le numéro du chapitre.

PROGRAMME BIBLIOTHEQUE

```
10 CLS : SCREEN 1,0,0 : ATTRB 1,1
20 LOCATE 8,1,0 : PRINT"BIBLIOTHEQUE" : ATTRB 0,0
30 LOCATE 2,4 : COLOR 0,3 : PRINT"1"; : COLOR 3,0
: PRINT" - Romans policiers"
40 LOCATE 2,7 : COLOR 0,3 : PRINT"2"; : COLOR 3,0
: PRINT" - Science Fiction"
50 LOCATE 2,10 : COLOR 0,3 : PRINT"3"; : COLOR 3,0
 : PRINT" - Romans historiques"
60 LOCATE 2,13 : COLOR 0,3 : PRINT"4"; : COLOR 3,0
 : PRINT" - Romans d'aventures"
70 LOCATE 2,16 : COLOR 0,3 : PRINT"5"; : COLOR 3,0
 : PRINT" - Livres d'enfants"
80 LOCATE 2,19 : COLOR 0,3 : PRINT"6"; : COLOR 3,0
 : PRINT" - Bandes dessinees
90 LOCATE 0,0 : INPUT"",A$
100 Y=CSRLIN-1
110 NL=(X-4)/3+1
```

Chapitre 6

LES CARACTERES UTILISATEURS

Comme nous l'avons vu dans le paragraphe précédent, l'écran est constitué de 1 000 cases réparties en 40 colonnes et 25 lignes. Dans chaque case, on peut faire apparaître un caractère, lettre ou symbole alphanumérique, signe de ponctuation...

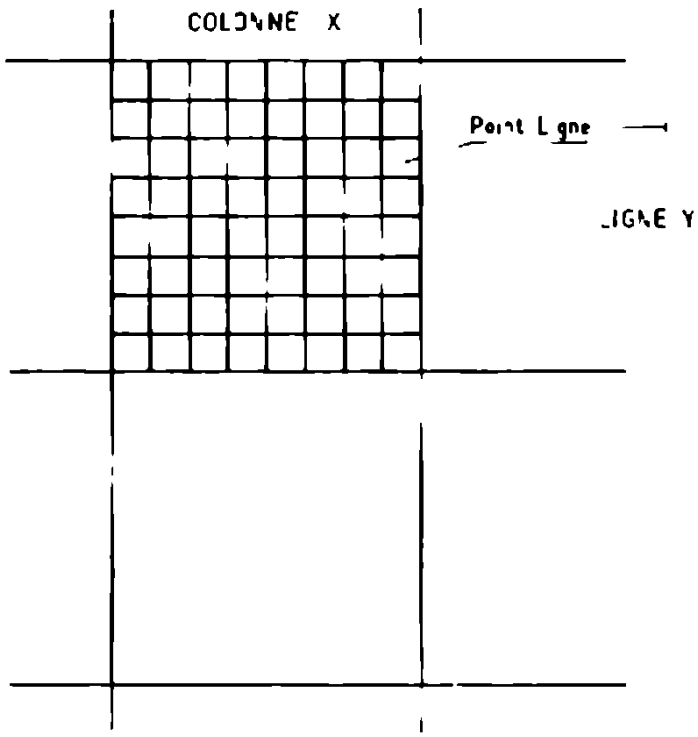


Fig. 6-1

Comment est donc obtenu le dessin d'un caractère sur l'écran ? Eh bien, chaque case est divisée en un quadrillage de 8 × 8 (fig. 6-1), soit 64 points lumineux que l'on appellera points ligne (PL). Nous savons que chaque case est définie par une couleur de fond et une couleur de caractère, c'est-à-dire que chaque PL (1) peut être allumé dans la couleur de fond ou dans la couleur du caractère. Nous adopterons le système suivant :

– Lorsqu'un PL sera allumé dans la couleur du fond, nous dirons qu'il est à zéro (Ø).

– Lorsqu'un PL sera allumé dans la couleur du caractère, nous dirons qu'il est à un (1).

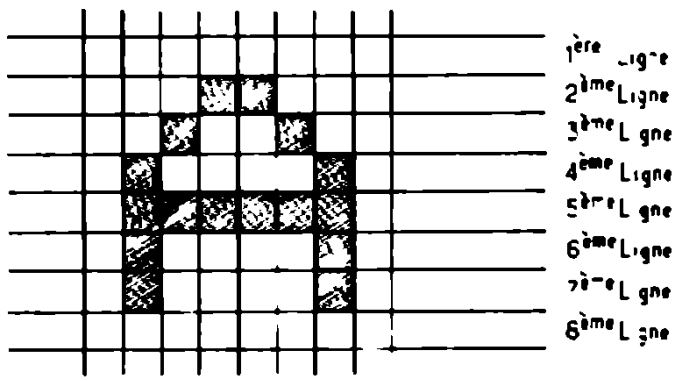


Fig. 6-2

Comment est constituée la lettre A majuscule qui apparaît sur l'écran ? Comme on le constate sur la figure 6-2, certains PL sont à Ø et d'autres à 1. Tout d'abord, un cadre de 1 PL de large est défini autour du caractère. Afin que les lettres qui se suivent à une case ne se touchent pas. La première ligne de 8 PL (que l'on appellera GPL : groupe de points ligne) ne contient que des PL à Ø, ce que l'on représente comme ceci :

première ligne = Ø Ø Ø Ø Ø Ø Ø Ø

La deuxième ligne voit 2 PL à 1 former la partie supérieure de la lettre A, donc :

deuxième ligne = Ø Ø Ø 1 1 Ø Ø Ø

(1) Les Anglo-Saxons disent « pixel ».

Finalement la lettre A est donc codée comme ceci :

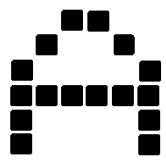
```
0 0 0 0 0 0 0 0
0 0 0 1 1 0 0 0
0 0 1 0 0 1 0 0
0 1 0 0 0 0 1 0
0 1 1 1 1 1 1 0
0 1 0 0 0 0 1 0
0 1 0 0 0 0 1 0
0 0 0 0 0 0 0 0
```

Or, dans le système de numération binaire (qui n'utilise que des 0 et des 1), on compte selon le code suivant :

Valeur décimale	Valeur binaire codée sur un octet (8 bits)
0	0 0 0 0 0 0 0 0
1	0 0 0 0 0 0 0 1
2	0 0 0 0 0 0 1 0
3	0 0 0 0 0 0 1 1
4	0 0 0 0 0 1 0 0
5	0 0 0 0 0 1 0 1
6	0 0 0 0 0 1 1 0
7	0 0 0 0 0 1 1 1
8	0 0 0 0 1 0 0 0
etc.	

Donc un 1 dans la colonne de droite vaut 1, alors qu'un 1 dans la deuxième colonne vaut 2, puis 4 dans la troisième, 8 dans la quatrième, 16... 32... 64... 128 dans la colonne de gauche.

On peut donc représenter le caractère A par les valeurs décimales de chacun des GPL qui le composent, soit :



=

```
  0
 24      16 + 8
 36      32 + 4
 66      etc.
126
 66
 66
  0
```

Ce que l'on peut écrire :

caractère (A) = 0, 24, 36, 66, 126, 66, 66, 0

1. Exemple introductif

Notre micro-ordinateur contient déjà un jeu de caractères figé sur lequel nous ne pouvons intervenir, mais son Basic va nous permettre de créer nos propres caractères, et ce, jusqu'à 128 caractères utilisateurs au maximum.

Supposons que nous ayons besoin de définir des lettres de l'alphabet grec, ou bien un message en chinois ou en arabe.

Par exemple :

La lettre q en arabe (qâf) s'écrit ق . Créons ce caractère.

Pour cela, dessinons-le dans un quadrillage 8 × 8 (fig. 6-3) ; notons 1 les PL constituant le caractère, et Ø ceux constituant le fond.

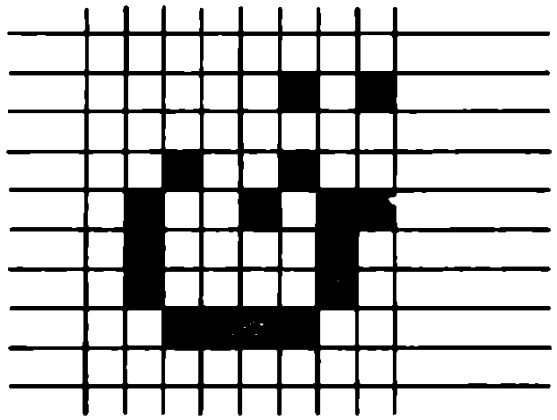


Fig. 6-3

2. Les instructions de définition

DEFGR$(X)

Cette instruction permet de définir le caractère utilisateur GR$ de numéro X. On définit ce caractère en donnant les 8 valeurs décimales représentant le code binaire des GPL constituant le caractère.

On commence par la ligne du haut et on termine par celle du bas.

Exemple : je veux créer la lettre A majuscule qui s'appellera GR$ (Ø). J'écris donc : DEFGR$(Ø) = Ø, 24, 36, 66, 126, 66, 66, Ø

Exemple : la lettre arabe q sera mon caractère GR$ (1) si j'écris :

DEFGR$(1) = Ø, 5, Ø, 36, 75, 66, 66, 6Ø

CLEAR,,N

Cette instruction est nécessaire pour réserver de la place en mémoire afin d'y ranger les valeurs définissant chaque ligne de chaque caractère utilisateur.

La valeur N indique combien de caractères vont être créés par la suite. L'instruction CLEAR,,N doit donc précéder toute définition de GR$.

Le nombre d'octets réservés en mémoire sera de 8 × N. On peut créer au maximum 128 caractères, numérotés de Ø à 127. Une fois créé, un caractère se manipule comme une chaîne. On peut donc afficher son caractère en tapant PRINT GR$(X).

On peut accoler deux caractères par concaténation...

Exemple : A$ = GR$(Ø) + GR$(1)

A$ devient un nouveau caractère de 2 cases de longueur.

3. Applications

– Rangement et pointage des GR$

Lorsque par l'instruction CLEAR,,N on réserve de la place pour définir N caractères, le moniteur lui réserve en mémoire N * 8 cases pour y ranger les valeurs décimales de définition des GR$.

Un pointeur, USERAF, d'adresse &H2070, &H2071, contient en permanence l'adresse du haut de la pile de rangement.

Exemple : En faisant CLEAR,,1
puis DEFGR$(Ø) = 1, 2, 3, 4, 5, 6, 7, 8

le moniteur range en mémoire :

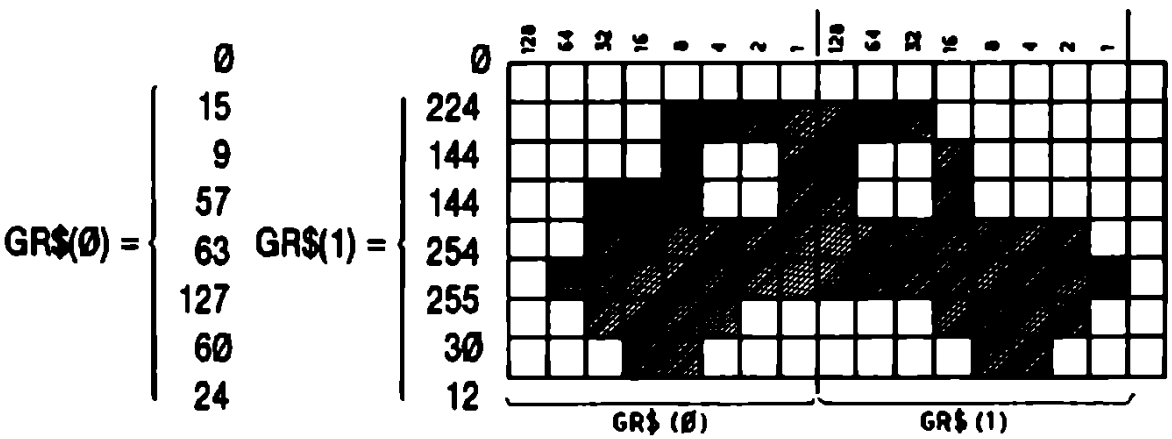
Adresse RAM	Contenu
AD	8
AD+1	7
AD+2	6
AD+3	5
AD+4	4
AD+5	3
AD+6	2
AD+7	1

et USERAF contient AD (2 octets).

Le programme suivant vous permet de vérifier le contenu de la plle.

```
10 CLS : SCREEN 2,0,0
20 CLEAR ,,1
30 DEFGR$(0) = 1,2,3,4,5,6,7,8
40 FOR N=0 TO 7
50      AD = PEEK(&H2070)*256+PEEK(&H2071)+N
60      PRINT AD,PEEK(AD)
70      NEXT
80 END
```

– Dessin et déplacement d'une automobile




```
10 CLS : SCREEN 7,0,0 : ATTRB 1,1
20 CLEAR ,,2
30 DEFGR$(0) = 0,15,9,57,63,127,60,24
40 DEFGR$(1) = 0,224,144,144,254,255,30,12
50 V$ = " "+GR$(0)+GR$(1)
60 FOR X=0 TO 32
70      LOCATE X,12
80      PRINTV$
90      NEXT
100 CLS : GOTO 60
110 END
```

– Sauvegarde des GR$ sous forme de fichier

On peut sauvegarder les GR$ sur la cassette, sous forme de fichier binaire (voir plus loin ce chapitre), à l'aide du programme ci-dessous :

```
10 CLEAR ,,2
20 DEFGR$(0) = 0,15,9,57,63,127,60,24
30 DEFGR$(1) = 0,224,144,144,254,255,30,12
40 AB = PEEK(&H2070)*256 + PEEK(&H2071)
50 AH = AB + 2*8
60 SAVEM"CASS:AUTO",AB,AH,AB
70 END
```

On peut ensuite le recharger à sa place à l'aide des lignes de programme ci-dessous :

```
10 CLEAR ,,2
20 LOADM"CASS:AUTO"
```

Si le nombre des GR$ est X, on écrit évidemment.

```
50 AH = AB + X*8
```

Remarque : Lorsque l'on a défini un certain nombre de GR$, on peut, par la suite, les utiliser comme des CHR$

Exemple :

PRINT CHR$ (128 + X) est équivalent à

PRINT GR$(X)

Donc, les CHR$(128) → CHR$(255) représentent les GR$(0) → GR$(127).

– Création de GR$

Afin de vous aider à fabriquer vos dessins, donc vos GR$, voici un petit programme qui va dans ce sens.

Il fait apparaître sur l'écran une matrice 8 × 8 correspondant à une grille caractère grossie 8 fois.

Vous pouvez y déplacer le curseur en tout sens à l'aide des 4 flèches de déplacement ↑ ↓ → ←.

Pour noircir une des 64 cases, appuyer sur [ENTREE] après avoir placé le curseur.

Pour effacer une des cases, ramener le curseur en bonne position et refaites [ENTREE]

Quand votre GR$(n) est terminé, appuyer sur [RAZ]

Vous verrez alors apparaître les 8 valeurs le constituant en bas de l'écran.

Puis une nouvelle grille apparaît à côté de la première, dans laquelle vous pourrez travailler comme ci-dessus.

En tout, vous pouvez donc faire un dessin contenant 15 GR$ contenus dans un rectangle de 5 × 3.

Alors, bon travail et beaux dessins.

```
10 CONSOLE 0,24 : SCREEN 2,0,0 : CLS
20 CLEAR,,1:DEFGR$(0)=255,255,255,255,255,255,255,
255:CLS:XD=0:DIMX(8,8)
30 FOR YD=0 TO 16 STEP 8
40 XC=XD:YC=YD
50 LOCATEXD,YD
60 FOR N=0 TO 8
70 LINE (8*XD,(8*YD)+(N*8))-(8*(XD+8)-1,8*YD+N*8)
80 LINE (8*XD+N*8,8*YD)-(8*XD+N*8,8*(YD+8)-1)
90 NEXTN
100 A$="":A$=INKEY$:IFA$=""GOTO100
110 XC=XC-(A$=CHR$(9) AND XC<XD+7)+(A$=CHR$(8) AND
 XC>XD)
120 YC=YC-(A$=CHR$(10) AND YC<YD+7)+(A$=CHR$(11) A
ND YC>YD)
130 LOCATEXC,YC
140 IFA$=CHR$(13)THEN C=POINT(XC*8+4,YC*8+4):IF C=
4 THEN X(XD+8-XC,YC-YD+1)=0:PSET(XC,YC)" ":XC=XC+1
:GOTO60 ELSE X(XD+8-XC,YC-YD+1)=1:PSET(XC,YC)GR$(0
):XC=XC+1:GOTO60
150 IFA$=CHR$(12)THEN GOTO160 ELSE GOTO100
```

```
160 LOCATE0,23:PRINT"GR$(";A;")=":
170 FOR L=1 TO 8
180 FOR C=1 TO8
190 N(L)=N(L)+X(C,L)*2^(C-1)
200 X(C,L)=0
210 NEXTC
220 PRINTRIGHT$(STR$(N(L)),LEN(STR$(N(L)))-1):",";
230 N(L)=0
240 NEXTL
250 PRINTCHR$(24):
260 A=A+1
270 IFXD+8<39 THEN XD=XD+8 :GOTO 40 ELSE XD=0:NEXT
 YD
280 END
```

Chapitre 7

BRANCHEMENTS, ITERATIONS ET SOUS-PROGRAMMES

1. Exemple introductif

Pour calculer le montant de l'impôt sur le revenu d'un salarié, on procède de la façon suivante :

1. Calcul du revenu imposable (RI) égal à 80 % × 90 % du revenu net + pensions et rentes – déductions.

2. Calcul du quotient familial (QF) égal à RI divisé par le nombre de parts NP.

3. Recherche de la tranche et application de la formule correspondante.

On se propose donc de réaliser un programme de calcul de l'impôt simplifié (pas de corrections éventuelles) qui permette de modifier facilement d'une année sur l'autre les minima et maxima du quotient familial de chaque tranche ainsi que les taux et coefficients de chacune (fig. 7-1).

Le programme fera appel à un sous-programme pour la lecture de ces coefficients, taux..., et à un autre sous-programme pour le calcul de l'impôt.

On peut donc décomposer le programme suivant l'organigramme de la page suivante (fig. 7-2).

QF minimum	QF maximum	Taux	Coefficient
0	12 620	0	0
12 620	13 190	0,05	631
13 190	15 640	0,10	1 290,5
15 640	24 740	0,15	2 072,5
24 740	31 810	0,20	3 309,5
31 810	39 970	0,25	4 900
39 970	48 360	0,30	6 898,5
48 360	55 790	0,35	9 316,5
55 790	92 970	0,40	12 106
92 970	127 860	0,45	16 754,5
127 860	151 250	0,50	23 147,5
151 250	172 040	0,55	30 710
172 040	195 000	0,60	39 312
195 000		0,65	49 062

Fig. 7-1. – Tableau des divers coefficients pour l'année 1982.
Impôt = (RI × Taux) – (Coeff. × NP) si $QF_{min} \leqslant QF \leqslant QF_{max}$.

2. Branchements inconditionnels

GOTO NL

Cette instruction force l'unité centrale à sauter à la ligne indiquée derrière GOTO.

ON VAR GOTO NL1, NL2,...

Suivant la valeur de la variable VAR, cette instruction force à la ligne NL1 si VAR = 1, NL2 si VAR = 2, etc.

Exemple :

```
10 PRINT"Quel chapitre choisissez-vous ? :".
20 INPUT X
30 ON X GOTO 100,200,300
```

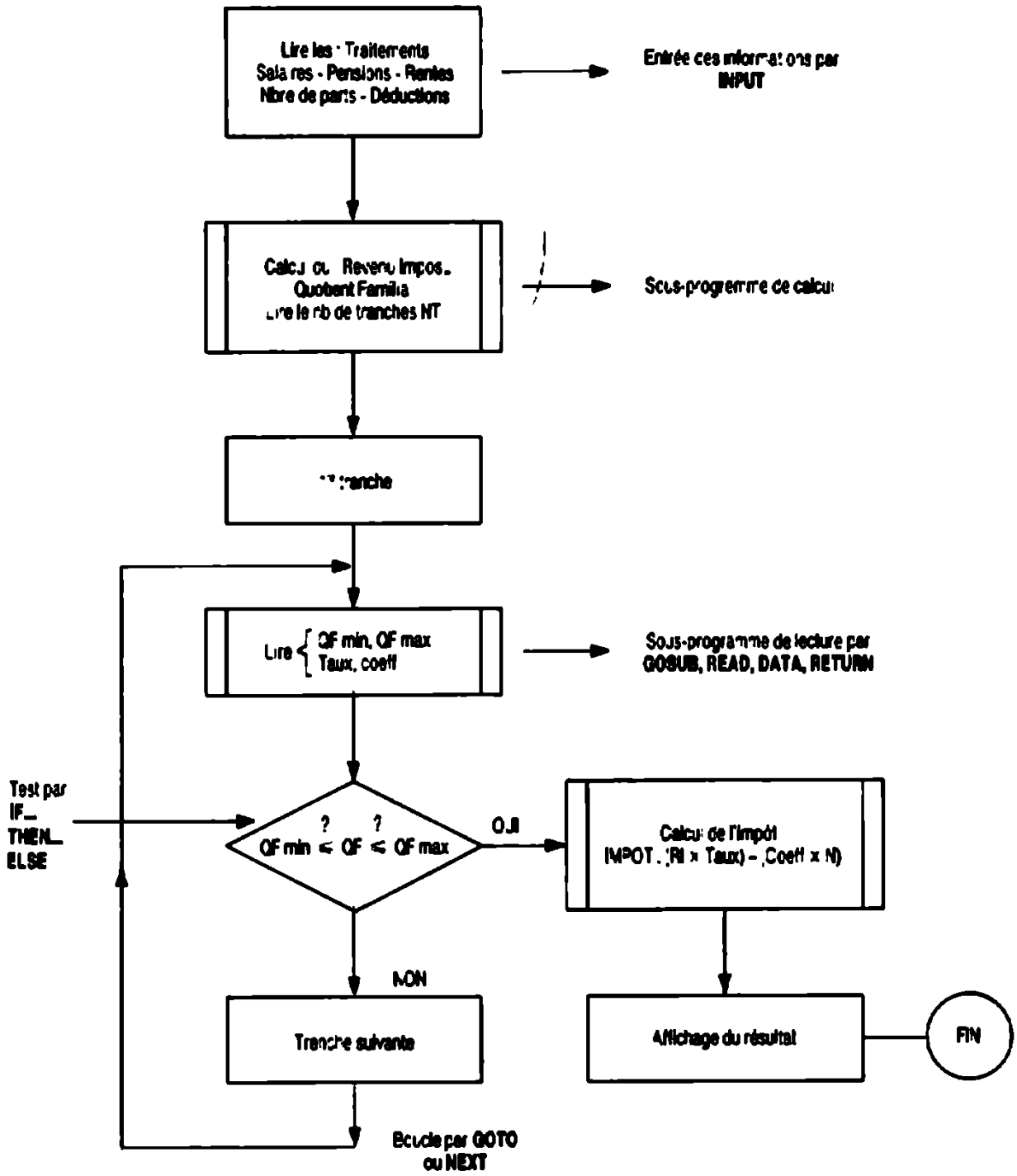


Fig. 7-2. – Organigramme.

3. Branchements conditionnels

IF... THEN... ELSE

Cette instruction permet de réaliser un test dont la réponse sera OUI ou NON. Si le test est vrai (réponse OUI), alors seules les instructions qui suivront THEN seront prises en compte.

Si le test est faux (réponse NON), alors seules les instructions suivant ELSE seront prises en compte.

Exemple :

```
10 INPUT "Quel est votre age :".AGE
20 IF AGE>10 THEN PRINT"Vous etes trop age." ELSE
GOTO 10
```

Les tests peuvent être multiples :

Exemple :

```
10 INPUT"Quel est votre age :".AGE
20 IF AGE>10 THEN GOTO 50 ELSE IF AGE <5 THEN PRIN
T"Vous etes trop jeune." ELSE GOTO 100
```

L'instruction THEN GOTO peut être remplacée par GOTO seule.

On peut omettre l'instruction ELSE. Dans ce cas, si le test est faux, le programme passe à la ligne suivante.

Exemple :

```
10 INPUT"Quel est votre age :".AGE
20 IF AGE>10 THEN PRINT"Vous etes trop vieux." : END
30 ...
```

test vrai

test faux

Dans l'exemple ci-dessus, on remarque que plusieurs instructions peuvent être lues en séquence, que le test soit vrai ou faux, à condition de les séparer par « : ».

Exemple :

```
10 INPUT"Quel est votre age :",AGE
20 IF AGE>10 THEN PRINT"Vous etes trop vieux." : E
ND ELSE IF AGE<5 THEN PRINT"Vous etes trop jeune."
 : END ELSE GOTO 100
30 ...
```

Opérateurs de test AND, OR, NOT, XOR, >, <, =, EQV, IMP

Le test suivant « Si A est plus grand que 2 et si B est plus petit ou égal à 3, alors aller ligne 100, sinon aller ligne 200 » peut s'écrire :

```
20 IF A>2 AND B<3 GOTO 100 ELSE GOTO 200
```

L'opérateur OR permet le test OU INCLUSIF.
L'opérateur NOT permet l'inversion du test.
L'opérateur XOR permet le test OU EXCLUSIF.
L'opérateur <> permet le test « différent de ».
L'opérateur EQV permet le test « équivalent à ».
L'opérateur IMP permet le test « implication ».

4. Les itérations

FOR...TO...STEP.../NEXT

Cet ensemble d'instructions permet la répétition de toute la partie du programme comprise entre l'ouverture de la boucle par FOR... TO... STEP et la fermeture par NEXT.

A l'ouverture de la boucle, on indique qu'une variable VAR doit être incrémentée d'une valeur initiale I jusqu'à une valeur finale F par pas de P.

Exemple : FOR X = 1 TO 10 STEP 1

Quand le pas est de 1, on peut omettre cette partie.

Exemple : FOR Z = 2 TO 16 STEP 2.

Lorsque le programme arrive à l'ouverture de la boucle, il affecte à la variable la valeur initiale, puis traite les lignes suivantes jusqu'à ce qu'il rencontre l'instruction NEXT.

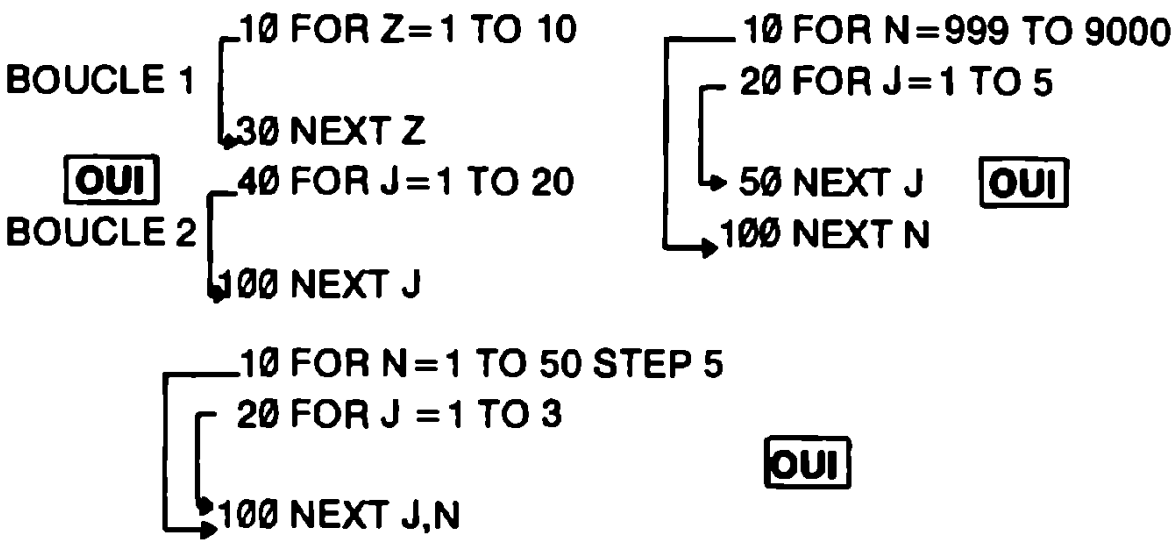
A cet instant, il incrémente la variable de la valeur du pas et traite les lignes suivantes avec cette nouvelle valeur, etc., jusqu'à ce que la variable atteigne la valeur finale. Les instructions sont lues une fois de plus, la variable est encore une fois incrémentée d'un pas, mais le programme ne revient plus en arrière et, au contraire, continue en séquence après l'instruction NEXT.

Exemple : on veut afficher la table de multiplication par 3, de 3 × 1 à 3 × 10.

```
10 CLS : PRINT"Table de multiplication par 3."
20 FOR N=1 TO 10
30      PRINT"3 *";N;"=";3*N
40      NEXT
50 END
```

On peut imbriquer plusieurs boucles ayant des variables différentes à condition d'indiquer (fig. 7-3) le nom de la variable derrière l'instruction NEXT.

Fig. 7-3.



Tous les schémas de programmes ci-dessus correspondent à des imbrications de boucles correctes. Par contre, le schéma de programme de la figure 7-4 n'est pas correct.

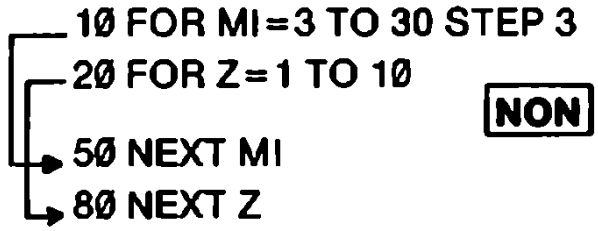
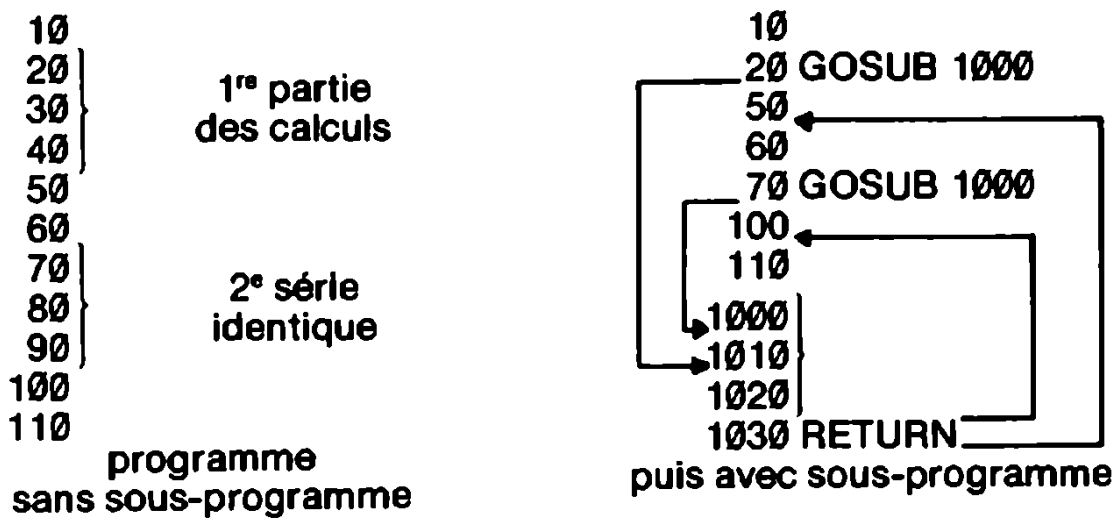


Fig. 7-4

5. *Branchements à des sous-programmes*

Lorsque, dans un programme, un calcul ou une série de calculs ou d'opérations sont effectués plusieurs fois à des lignes différentes, il est intéressant de faire de cette série de calculs ou d'opérations un sous-programme, que l'on appelle à partir du programme principal par l'instruction GOSUB suivie du numéro de la première ligne du sous-programme, et qui se termine par RETURN pour reprendre en séquence le programme principal.

Exemple :



GOSUB NL

Cette instruction dirige vers le sous-programme commençant à la ligne NL.

ON Var GOSUB NL1, NL2 :

Cette instruction agit comme ON GOTO, avec cette fois appel d'un sous-programme.

RETURN

Cette instruction termine nécessairement un sous-programme et permet le retour en séquence dans le programme principal.

D'un sous-programme on peut appeler un deuxième sous-programme, etc., sans limitation.

De même, d'un sous-programme on peut appeler ce sous-programme lui-même (récursivité).

Un sous-programme peut avoir plusieurs points d'entrée et plusieurs RETURN.

EXEC AD

Cette instruction permet d'aller réaliser un sous-programme écrit en langage 6809 et commençant à l'adresse AD.

Ce sous-programme doit commencer par sauvegarder la valeur du registre de pile S et du registre de page zéro DP (voir chapitre sur le 6809) pour les restituer avant de revenir au programme Basic.

Ce sous-programme doit se terminer par l'instruction Assembleur RTS qui est l'équivalent de RETURN mais en langage machine (code &H39).

Le moniteur existant en mémoire morte contient déjà un certain nombre de sous-programmes que l'on peut utiliser (voir annexe 5).

Cette instruction ne permet pas d'échanger de paramètre avec le sous-programme Assembleur.

CLEAR, AD

Cette instruction indique l'adresse maximum AD utilisable par le Basic.

Donc, à partir de AD+1, l'espace est réservé pour ranger un sous-programme en Assembleur.

```
1 '*****************************************
2 '*        CALCUL DE L'IMPOT              *
3 '*                                       *
4 '*****************************************
10 CLS : SCREEN 7.0,0
20 LOCATE 3,2,0 : ATTRB 1,1 : PRINT"CALCUL DE L'IM
POT" : ATTRB 0,0
30 LOCATE 0,6 : INPUT "Total des salaires : ",A
40 LOCATE 0,8 : INPUT"Frais reels si option : ",B
50 LOCATE 0,10 : INPUT"Deduction forfaitaire sup.
: ",D
60 LOCATE 0,12 : INPUT"Pensions, retraites, ... -1
0% : ",G
70 LOCATE 0,14 : INPUT"Charges a deduire : ",J
80 LOCATE 0,16 : INPUT"Nd de parts : ",N
90 GOSUB 1000
100 FOR X=1 TO NT
110 GOSUB 2000
120 IF QF>QFMI AND QF<=QFMA THEN GOSUB 3000 : LOCA
TE 3,2 : ATTRB1,1 : PRINT"IMPOT : F.";I1 : ATTRB 0
,0 : END ELSE NEXTX
130 DATA 14,0,12620,0,0,12620,13190,0.05,631,13190
,15640,0.1,1290.5,15640,24740,0.15,2072.5,24740,31
810,0.2,3309.5,31810,39970,0.25,4900,39970,48360,0
.3,6898.5,48360,55790,0.35,9316.5
140 DATA 55790,92970,0.4,12106,92970,127860,0.45,1
6754.5,127860,151250,0.5,22147.5,151250,172040,0.5
5,30710,172040,195000,0.6,39312,195000,999999,0.65
,49062
150 END
```

```
990 '
1000 '          CALCUL DU REVENU
1005 '          ================
1010 '
1020 IF B=0 THEN E=A*0.1+D ELSE E=B
1030 F = A-E
1040 H = 0.2*(F+G)
1050 RB = F+G-H : LOCATE 0,18 : PRINT"Revenu brut
: ":RB
1060 RI = RB-J : LOCATE 0,20 : PRINT"Revenu imposa
ble : ":RI
1070 QF = RI/N : LOCATE 0,22 : PRINT"Quotient fami
lial : ";QF
1080 READ NT
1990 '
2000 '          LECTURE D'UNE TRANCHE
2005 '          =====================
2010 '
2020 READ QFMI,QFMA,TAUX,COEFF
2030 RETURN
2990 '
3000 '          CALCUL DE L'IMPOT
3005 '          =================
3010 '
3020 I1 = (RI*TAUX) - (COEFF*N)
3030 RETURN
```

Chapitre 8

LES VARIABLES NUMERIQUES

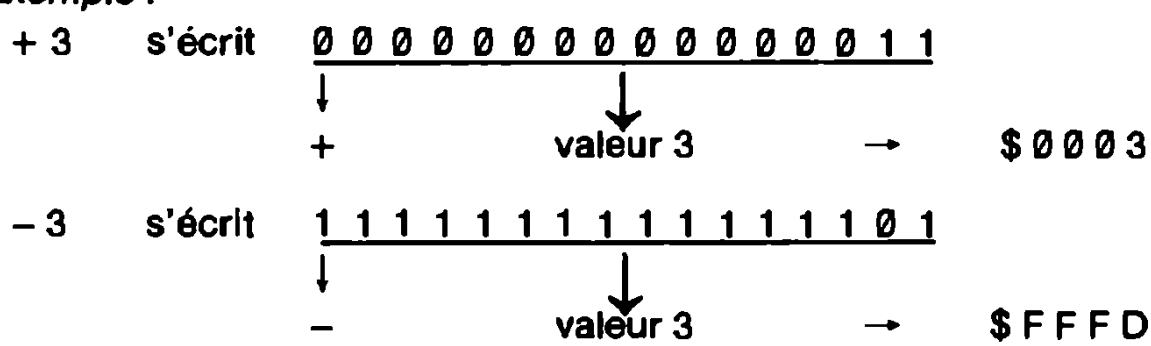
1. Représentation interne

Comme nous l'avons vu au chapitre 4 (*Les bases du langage*, page 27), les variables numériques peuvent être entières (%) ou simple précision (!), c'est-à-dire réelles.

Quel que soit le type de variable, sa valeur est rangée en mémoire selon une écriture binaire.

Les variables entières occupent chacune 2 octets, soit 16 bits par variable. Le bit de poids le plus fort est utilisé pour représenter le signe de la variable (0 si +, 1 si –). Si la variable entière est positive, les 15 autres bits donnent sa valeur en mode binaire (maximum : 2^{15} = 32768, donc de 0 à 32767). Si la variable entière est négative, les 15 autres bits donnent la valeur en mode complément à deux (donc de – 1 à – 32768).

Exemple :



Les variables réelles occupent 4 octets en mode virgule flottante. Ce mode virgule flottante permet une importante économie de place en mémoire, et fonctionne de la façon suivante :

Tout nombre est représenté par ses chiffres significatifs (la mantisse M) et un exposant E. La valeur du nombre N est alors :

$$N = (M + 0{,}5) * 2^{E-128}$$

L'exposant E est contenu en binaire dans le premier octet, les trois autres octets constituant la mantisse. Le bit de poids le plus fort de cette mantisse représente le signe du nombre.

Exemple :

N	81	00	00	00

mantisse positive M = 0

exposant E = $81 = 8 * 16 + 1 = 129

N = 0,5 * 2 = 1

N	82	00	00	00	N = 0,5 * 4 = 2
N	83	C4	00	00	

mantisse : bit de poids fort à 1 → N négatif

$M = (C-8) * 16^{-1} + 4 * 16^{-2} = 0{,}25 + 0{,}015625 = 0{,}265625$

exposant : E = $83 = 8 * 16 + 3 = 131

$N = -(0{,}265625 + 0{,}5)2^3 = -6{,}125$

Les variables et leurs valeurs sont rangées à la suite du programme Basic. Chaque variable est précédée d'une valeur qui précise son type.

Une variable non déclarée est considérée comme simple précision et occupe donc 4 octets (au lieu des 2 nécessaires ici). On voit donc l'intérêt de spécifier le type de variable.

On peut également définir ce type pour plusieurs variables à l'aide des instructions ci-dessous :

DEFINT A

Cette instruction indique que toutes les variables non spécifiées commençant par la lettre A sont des variables entières.

Exemple :

A	est une variable entière
ALG	est une variable entière
A!	est une variable simple précision
A et A!	sont considérées comme deux variables distinctes.

DEFSNG A-C

Cette instruction indique que toutes les variables non spécifiées commençant par les lettres A, B ou C sont des variables simple précision.

Il existe également, pour les chaînes de caractères, l'instruction DEFSTR.

2. *Les opérateurs arithmétiques*

Ces opérateurs sont, par ordre de priorité décroissante :

↑	Puissance	ex. : 2 ^ 2
–	Moins (signe)	ex. : – 3
* et /	Multiplication et division	ex. : 4 * 3/5
@	Division entière	ex. : 5 @ 2 = 2
MOD	Reste d'une division	ex. : 5MOD2 = 1
+ et –	Addition et soustraction	ex. : 4 + 3 – 8

L'élévation à la puissance se fait en mode shifté (touche jaune).

3. *Les opérateurs relationnels*

Ce sont les opérateurs classiques :

=	Egalité
<	Inférieur
<=	Inférieur ou égal
>	Supérieur
>=	Supérieur ou égal
<>	Différent de

Le résultat d'un test est une valeur binaire 0 ou – 1 suivant que le test est faux ou vrai. Cette valeur peut d'ailleurs être affectée à une variable et utilisée par la suite.

Exemple :

```
10 INPUT A,B,C,D : X = A+B <= C/D
20 IF X=0 THEN PRINT "A+B est plus grand que C/D."
30 PRINT "X = ";X
40 GOTO 10
```

4. *Les fonctions arithmétiques et algébriques*

ABS (Arg) Cette instruction calcule la valeur absolue de l'argument entre parenthèses et donne un résultat du même type que l'argument.

CINT (Arg) Cette instruction convertit l'argument en type entier.

COS (Arg) Cette instruction calcule le cosinus de l'argument exprimé en radians.

EXP (Arg) Cette instruction calcule l'exponentielle de l'argument.

FIX (Arg) Cette instruction donne la partie entière signée de l'argument. Il ne s'agit pas d'une conversion, et le résultat est encore rangé en mémoire comme une valeur réelle.

INT (Arg) Cette instruction donne la valeur entière immédiatement inférieure ou égale à l'argument.

LOG (Arg) Cette instruction calcule le logarithme népérien de l'argument.

RND (Arg) Cette instruction permet d'obtenir une valeur pseudo-aléatoire, réelle, comprise entre 0 et 1 inclus. Si l'argument est négatif, plusieurs appels de cette instruction donnent la même valeur, sinon la valeur change à chaque fois.

SGN (Arg) Cette instruction donne le signe de l'argument.

SIN (Arg) Cette instruction calcule le sinus de l'argument (en rad.).

SQR (Arg) Cette instruction calcule la racine carrée de l'argument.

TAN (Arg) Cette instruction calcule la tangente de l'argument exprimé en radians.

5. *Applications*

• Conversion d'un nombre entier inférieur à 256 en sa valeur binaire

Ce nombre sera codé sur 8 bits que l'on appellera B(Ø), B(1),... B(7) en partant du poids faible jusqu'au poids fort. Donc :

$$N = B(7) * 2^7 + B(6) * 2^6 + ... + B(Ø) * 2^0$$

On en déduit donc que :

– B(7) est la partie entière de la division de N par 2^7
– B(6) est la partie entière de la division de N – B(7) * 2^7, par 2^6, etc.

La structure du programme sera donc la suivante :

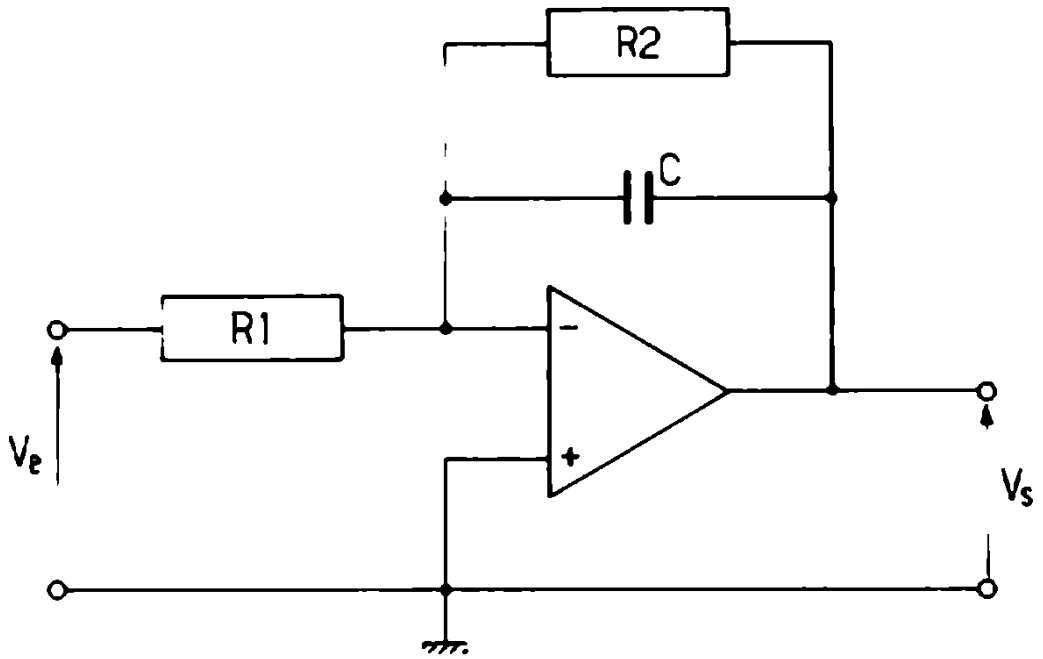
1. Introduire la valeur de N.
2. Ouvrir une boucle de J = 7 à J = Ø
   – calculer B(J) = INT (N/(2 ↑ J))
   – poser N = N – B(J) * 2^J.
3. Afficher le résultat. Ce qui conduit au programme suivant :

```
10 '       **********************
20 '       *                    *
30 '       * CONVERSION DEC-BIN *
40 '       *                    *
50 '       **********************
60 '
100 INPUT "Donner la valeur du nombre : ".N
110 FOR J=7 TO 0 STEP -1
120     B(J) = INT(N/2^J)
130     N = N - B(J)*2^J
140     PRINTB(J):
150     NEXT J
160 PRINT
170 GOTO 100
180 END
```

On constate que le programme ci-dessus a été écrit en décalant de quelques caractères l'ensemble des instructions de la boucle FOR... NEXT, afin de rendre le programme plus lisible. On peut parler alors *d'écriture STRUCTUREE.*

• Tracé de la courbe de réponse en fréquence d'un filtre R-C.

Le but de cette étude est d'analyser le comportement d'un filtre actif lorsque l'on fait varier la valeur de l'un des composants. Soit le filtre ci-dessous :



Le gain de ce filtre est :

$$A_V = \frac{V_S}{V_e} = - \frac{R_2/R_1}{\sqrt{1 + R^2_2 C^2 \omega^2}}$$

avec $\omega = 2.\pi.f$ f = fréquence en Hz.

On se propose d'étudier l'influence de la valeur de C dans la réponse en fréquence de ce filtre passe-bas du 1er ordre. On supposera $R_2 = 100 \cdot R_1 = 100$ kΩ.

La fréquence f prendra les valeurs multiples de 10, 20, 50 dans une gamme allant de 10 Hz à 500 kHz.

La courbe représentera le rapport $|A_V/A_{VO}|$ en décibels, soit :

$$20 * \log_{10}\left[\frac{A_V}{R_2/R_1}\right] = 20 * \log_{10}\left[\frac{1}{\sqrt{+R^2_2C^2\omega^2}}\right]$$

$$= -10 * \log_{10}(1 + R^2_2C^2\omega^2)$$

en fonction de la fréquence f. (On portera en fait la valeur de $\log_{10} f$ en abscisse.)

On devra également faire apparaître sur l'écran la valeur de la fréquence de coupure $f_o = \frac{1}{2\pi R_2 C}$.

Ce programme sera étudié en deux parties :
- 1re partie : calcul des coordonnées des points de la courbe.
- 2e partie : tracé de la courbe.

La deuxième partie sera étudiée dans les chapitres 13 et 14 consacrés aux dessins et graphiques.

On rappelle que $\text{Log}_{10} X = \text{Log}_e X * \text{Log}_{10} e = 0{,}43429419 * \text{LNX}$

LNX

Donc $G_{dB} = -4{,}3429419 * \text{LN}(1 + (R_2 * C * 2 * \pi * f) \uparrow 2)$

On en déduit le programme suivant :

```
10 '          ************************
20 '          *                      *
30 '          * REPONSE D'UN FILTRE *
40 '          *                      *
50 '          ************************
60 '
70 '
80 '------------CALCUL DU GAIN------
90 '
95 CONSOLE 0,24 : SCREEN 2,0,0 :CLS
100 A$ = "###.##^" : B$ = "##" : C$ = "##^"
110 DIM G(15),F(15) : E = 4.3429419 : P = 3.1415927 : R2
 = 1 E 5
120 INPUT "Donner la valeur de C : ",C
130 FO = 1/(2*P*R2*C) : CLS : PRINT"FO = ";INT(FO);"Hz"
```

```
: PRINT
140 FOR I=1 TO 15
150     READ F(I)
160     G(I) = -E*LOG(1+(R2*C*2*P*F(I))^2)
170     PRINT"F(": : PRINTUSING B$;I; : PRINT") =": : PR
INTUSING C$;F(I); : PRINT"    G(": : PRINTUSING B$;I; :
PRINT") =": : PRINTUSING A$;G(I)
180     NEXT I
181 DATA 10,20,50,100,200,500,1E3,2E3,5E3,1E4,2E4,5E4,1E
5,2E5,5E5
```

On remarquera que l'instruction REM peut être remplacée par une apostrophe.

Les résultats sont reproduits ci-dessous :

```
F( 1) = 1E+01     G( 1) =-17.08E-05
F( 2) = 2E+01     G( 2) =-68.51E-05
F( 3) = 5E+01     G( 3) =-42.83E-04
F( 4) = 1E+02     G( 4) =-17.11E-03
F( 5) = 2E+02     G( 5) =-68.04E-03
F( 6) = 5E+02     G( 6) =-40.88E-02
F( 7) = 1E+03     G( 7) =-14.45E-01
F( 8) = 2E+03     G( 8) =-41.15E-01
F( 9) = 5E+03     G( 9) =-10.36E+00
F(10) = 1E+04     G(10) =-16.07E+00
F(11) = 2E+04     G(11) =-22.01E+00
F(12) = 5E+04     G(12) =-29.95E+00
F(13) = 1E+05     G(13) =-35.96E+00
F(14) = 2E+05     G(14) =-41.98E+00
F(15) = 5E+05     G(15) =-49.94E+00
```

Chapitre 9

LES CHAINES DE CARACTERES

1. Représentation interne

Lorsque, dans un programme, on définit une variable chaîne, automatiquement, en fin de programme, sont rangés en mémoire :

- le séparateur indiquant une déclaration de chaîne,
- le nom de la variable.

Puis, sur 3 octets, on trouve :

- la longueur (nb total de caractères) de la variable,
- l'adresse où elle est définie, codée sur 2 octets.

La longueur de la variable, étant codée sur un octet, ne peut donc dépasser 255 caractères.

2. Opérateur concaténation

Si l'on veut fabriquer une chaîne de plus de 255 caractères, une méthode existe qui consiste à découper cette chaîne en autant de sous-chaînes que nécessaire puis à utiliser la somme (ou concaténation) de celles-ci.

Exemple : si longueur de A$ = 200
si longueur de B$ = 100
on peut toutefois écrire : PRINT A$ + B$

Le signe + représente la concaténation (mise bout à bout) des deux chaînes A$ et B$.

3. Opérateurs relationnels

Les opérationnels relationnels =, <, >, sont utilisables avec les chaînes de caractères.

Les deux premières lettres sont comparées, puis, si elles sont identiques, les deux suivantes...

Une chaîne est inférieure à une autre dès que l'une des deux lettres de la comparaison est plus petite (c'est-à-dire plus près du A) que l'autre, ou si, à égalité, l'une des chaînes est plus courte.

Exemple :

« AFRIQUE » > « AFRICAIN »

car Q > C

par contre : « AFRICAIN » < « AFRICAINE », car plus court.

Ces opérateurs permettront de réaliser facilement un rangement par ordre alphabétique.

Exemple : Rangement et mise par ordre alphabétique de 8 variables.

```
10 '       ********************************
20 '       *                              *
30 '       * PROGRAMME DE TRI ALPHABE.    *
40 '       *                              *
50 '       ********************************
60 '
70 '
80 '--------LECTURE DES DONNEES--------
90 '
100 FOR N=1 TO 8
110     READ A$(N)
120     PRINT A$(N)
130     NEXT
140 '
150 '
160 '------COMPARAISON DES CHAINES------
170 '
180 B$(1) = A$(1)
190 FOR I=2 TO 8
200     FOR J=1 TO I-1
210         IF A$(I)<=B$(J) THEN GOSUB 400 : GOTO
240
```

```
220           NEXTJ
230      B$(I) = A$(I)
240 NEXTI
250 '
260 '
270 '------AFFICHAGE DES RESULTATS----
280 '
290 '
300 PRINT : PRINT : PRINT
310 FOR S=1 TO 8
320      PRINTB$(S)
330      NEXTS
340 END
350 '
360 '
370 '----SOUS-PROGRAMME DE PERMUTATION--
380 '
390 '
400 FOR N=I TO J+1 STEP -1
410      B$(N) = B$(N-1)
420      NEXTN
430 B$(J) = A$(I)
440 RETURN
450 '
460 '----RANGEMENT DES CHAINES--------
470 '
480 '
490 DATA CHARLES,STEPHANE,ANDRE,MICHEL,ALPHONSE,ET
IENNE,ANATOLE,PAUL
```

Résultat :

```
ALPHONSE
ANATOLE
ANDRE
CHARLES
ETIENNE
MICHEL
PAUL
STEPHANE
```

4. *Les instructions de chaînes*

ASC (A$)

Cette instruction renvoie, en décimal, le code ASCII du premier caractère de la chaîne A$.

CHR$ (Arg)

Cette instruction permet d'obtenir le caractère dont le code ASCII est donné en décimal par l'argument. (Voir, plus loin, police des CHR$.)

CLEAR Arg

Cette instruction est nécessaire pour réserver en mémoire plus de 300 caractères de chaîne.

En résumé, l'instruction CLEAR a trois arguments utilisables simultanément et séparés par une virgule.

Le premier réserve de la place pour les chaînes.
Le second pour un sous-programme en assembleur.
Le troisième pour les caractères GR$.

Exemple :

CLEAR 400	&H8000	5
↓	↓	↓
400 octets chaîne	Fin du Basic et des variables	5 . GR$

FRE (A$)

Cette instruction renvoie le nombre d'octets restant disponibles pour les chaînes.

INSTR (Arg, A$, B$)

Cette instruction permet de rechercher la position de la sous-chaîne B$ dans la chaîne A$, à partir du caractère de rang indiqué par l'argument s'il est présent.

LEFT$ (A$, Arg)

Cette instruction permet d'obtenir une chaîne réalisée en ne conservant que les caractères à gauche de A$ en nombre égal à l'argument.

LEN (A$)

Cette instruction renvoie la valeur décimale de la longueur de la chaîne A$.

MID$ (A$, Arg, L)

Cette instruction permet d'extraire la chaîne de longueur L réalisée en ne conservant que les caractères de A$ à partir du rang donné par l'argument.

RIGHT$ (A$, Arg)

Cette instruction permet d'obtenir une chaîne en ne conservant que les caractères à droite de A$ en nombre égal à l'argument.

STR$ (Arg)

Cette instruction fournit une chaîne représentant le nombre donné par l'argument.

VAL (A$)

Cette instruction donne la valeur décimale du nombre chaîne A$.

5. Police de caractères – CHR$ (1)

De Ø à 31 (valeurs décimales), on a l'équivalence suivante :

Ø	NUL	12	RAZ
1		13	ENTREE
2	STOP	14	Semi-graphique
3		15	Alphanumérique
4		16	
5		17	Curseur clignotant
6		18	Répétition
7	BEEP	19	
8	←	20	Arrêt du curseur
9	→ déplacement	21	
10	↓ du curseur	22	Mode accentué
11	↑	23	

(1) Voir annexe 7.

24	Efface la fin de la ligne	28	INS
25		29	EFFACE
26		30	HOME
27	ESC	31	US

De 32 à 127, le programme ci-dessous permet d'obtenir l'équivalence nombre décimal ⇆ caractère.

```
10 '          **************************
20 '          *                        *
30 '          * POLICE DE CARACTERES   *
40 '          *                        *
50 '          **************************
60 '
70 '
80 FOR N=32 TO 127 STEP 4
90      PRINTN;" "+CHR$(N);TAB(10);
100     PRINTN+1;" "+CHR$(N+1);TAB(20);
110     PRINTN+2;" "+CHR$(N+2);TAB(30);
120     PRINTN+3;" "+CHR$(N+3)
130     NEXT
140 END
```

32		33	!	34	"	35	#
36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+
44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3
52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;
60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C
68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K
76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S
84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[

92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c
100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k
108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s
116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{
124	¦	125	}	126	~	127	

Remarque : dans les liaisons avec différentes imprimantes, certains codes ci-dessus (de Ø à 31) peuvent avoir des effets spéciaux.

Par exemple, avec l'imprimante à impact PR90.080, l'impression de CHR$(14) provoque le passage en mode double largeur et CHR$(15) ramène au mode normal.

6. Codes spéciaux

Les caractères GR$ définis par l'utilisateur peuvent être appelés par l'instruction CHR$ en respectant la correspondance suivante :

GR$ (N) = CHR$ (128 + N)

Exemple :

GR$(Ø) = CHR$(128)
GR$(1) = CHR$(129)...

Les codes des accents sont les suivants :

Accent grave 65 — Accent aigu 66
Accent circonflexe 67 — Tréma 72
Cédille 75

Exemple : Pour écrire « été », il faut taper :

```
PRINT CHR$(22) + CHR$(66) + CHR$(101) + « t » + CHR$(22)
      + CHR$(66) + « e »
```

Chapitre 10

LES TABLEAUX DE VARIABLES

1. Exemple introductif

Supposons que nous ayons rangé sur la cassette toutes les données d'un répertoire alphabétique, soit 50 noms de personnes, 50 prénoms, 50 adresses et 50 numéros de téléphone.

L'ensemble du fichier représente 200 variables et, si nous voulons pouvoir travailler avec toutes ces variables simultanément, il faut pouvoir les ranger en mémoire sous des noms différents.

D'autre part, il est souhaitable de pouvoir traiter ces variables dans des boucles de 1 à 50.

La solution idéale consiste donc à *indicer* chaque variable, c'est-à-dire à utiliser le même nom de variable suivi d'un numéro entre parenthèses.

Exemple :

```
NOM$(1)  = « DURAND »     PRE$(1)  = « JEAN »...
NOM$(2)  = « DUBOIS »     PRE$(2)  = « GEORGES »...
NOM$(3)  = « LEPIED »     PRE$(3)  = « CLAUDE »...
   ''                        ''
   ''                        ''
   ''                        ''
NOM$(50) = « MAHUT »      PRE$(50) = « MICHEL »...
```

2. Dimensionnement – Occupation

Le micro-ordinateur MO5 doit réserver en mémoire la place nécessaire au rangement des variables.

S'il rencontre, dans un programme, une variable indicée, il réserve automatiquement la place nécessaire au stockage de 11 variables du même type, numérotées (indicées) de Ø à 1Ø.

Exemple :

```
30 NOM$( 1 )="DURAND"
```

⇒ l'ordinateur réserve la place pour
NOM$(Ø), NOM$(1), NOM$(2)... NOM$(1Ø)

Si l'indice dépasse 1Ø, il faut dimensionner la variable.

DIM

Cette instruction réserve en mémoire la place nécessaire au rangement des variables déclarées.

Exemple :

```
10 DIM NOM$(50),PRE$(50),ADR$(50),TEL$(50)
```

Les instructions ci-dessus réservent la place pour 2Ø4 variables, NOM$(Ø)... NOM$(5Ø), PRE$(Ø)... PRE$(5Ø)...

Dans l'exemple précédent, on aurait pu identifier chaque personne par une fiche. Chaque fiche est numérotée de 1 à 5Ø.

Les variables sur les fiches sont repérées par des numéros.

S'il s'agit d'un nom,	le numéro est 1
S'il s'agit d'un prénom,	le numéro est 2
S'il s'agit d'une adresse,	le numéro est 3
S'il s'agit d'un numéro de téléphone,	le numéro est 4

Donc le prénom de la fiche 22 sera identifié par :

FIC$ (22,2) = « PAUL »

Le nom de la fiche 21 par : FIC$ (21,1) = « LABRANCHE »

On définit ainsi *un tableau* de variables à deux dimensions.

FICHE (X,Y)	Y=1	Y=2	Y=3	Y=4
X=1	DURAND	JEAN	20 rue des Prés	0340619
X=2	DUBOIS	GEORGES		
X=3	LEPIED			
X=4				

Fig. 10-1. – Tableau à deux dimensions.

Ce tableau doit être dimensionné par l'instruction :

DIM FIC$ (50,4)

On peut ainsi théoriquement définir des tableaux ayant jusqu'à 255 indices, chaque indice pouvant aller jusqu'à 32767, si la place en mémoire le permet.

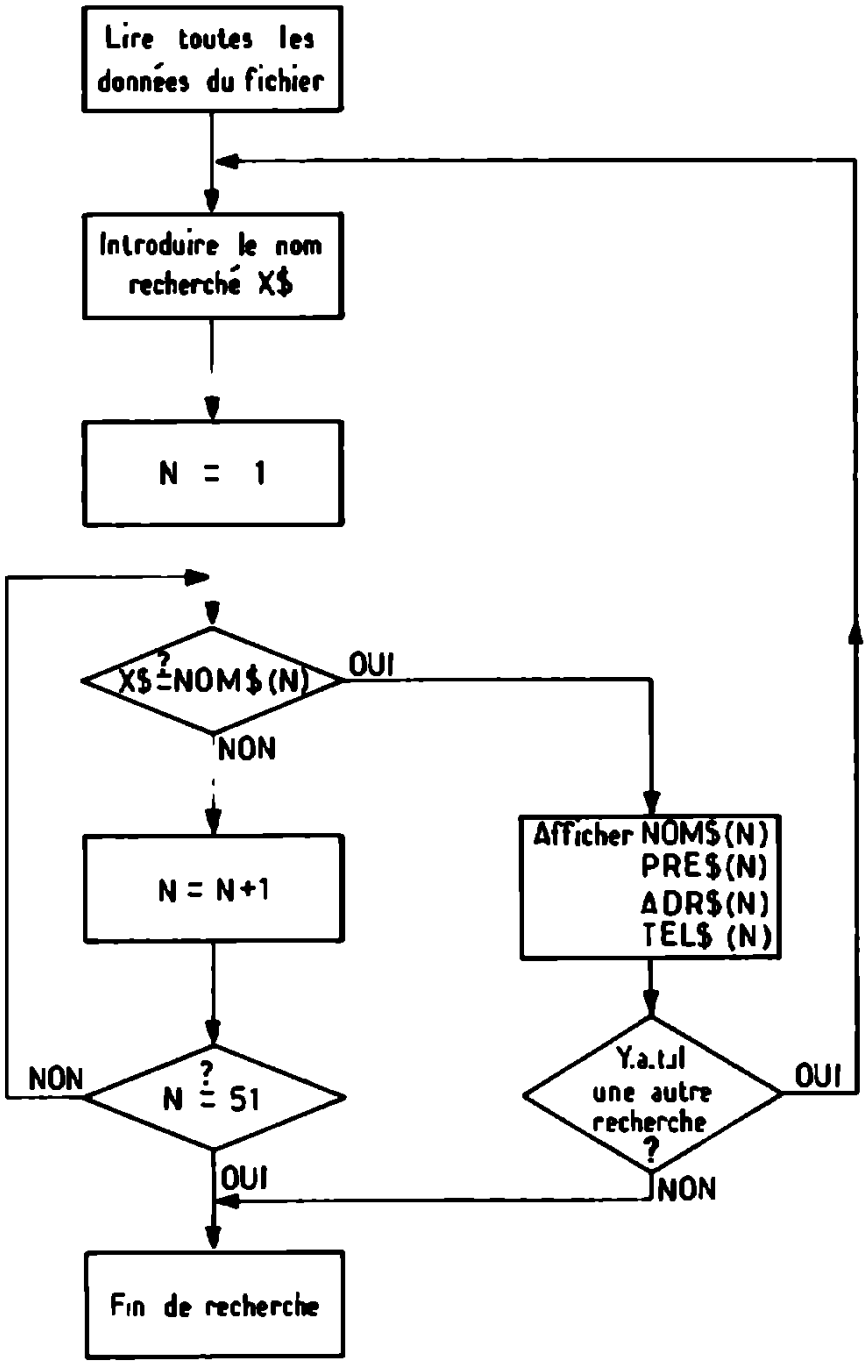
Les valeurs de ces variables sont ou bien définies dans le programme, ou bien rangées en fin de mémoire, juste avant les GR$.

3. Application

Supposons que le fichier existe déjà sur cassette et examinons le problème qui consiste, à partir d'un nom donné par l'opérateur, à rechercher les coordonnées correspondantes.

Cette méthode évite, dans le cas d'une recherche répétitive, de rembobiner la cassette pour relire à chaque fois le fichier, ce qui prend beaucoup trop de temps. Ici le fichier n'est lu qu'une seule fois en début de programme et toutes les variables indicées sont rangées en mémoire.

Cette méthode n'est plus utile avec un lecteur de disquette.


Lire toutes les données du fichier
Introduire le nom recherché X$
N = 1
X$? = NOM$(N)
OUI
NON
N = N+1
N ? = 51
NON
OUI
Afficher NOM$(N)
PRE$(N)
ADR$(N)
TEL$ (N)
Y.a.t.il une autre recherche ?
OUI
NON
Fin de recherche

```
10 '        ****************************
20 '        *                          *
30 '        * RECHERCHE D'UNE ADRESSE  *
40 '        *                          *
50 '        ****************************
60 '
70 '
80 '
90 DIM NOM$(50).PRE$(50),ADR$(50),TEL$(50)
100 OPEN"I",#2,"REPER"
110 FOR N=1 TO 50
120      INPUT#2,NOM$(N),PRE$(N),ADR$(N),TEL$(N)
130      NEXT N
140 INPUT "Nom recherche : ",N$
150 N = 1
160 IF N$=NOM$(N) GOTO 200
170 N = N+1 : IF N = 51 GOTO250
180 GOTO 160
190 END
200 PRINTNOM$(N)+" "+PRE$(N)
210 PRINTADR$(N)
220 PRINT"                TEL : "+TEL$(N)
230 PRINT : PRINT : PRINT"Cherchez-vous quelqu'un
d'autre ?"
240 A$ = "" : A$ = INKEY$ : IF A$="" GOTO 240 ELSE
 IF A$="O" GOTO 140 ELSE CLOSE#2 : END
250 PRINT"Erreur. Ce nom n'est pas dans le reperto
ire." : CLOSE :END
```

Chapitre 11

LES INSTRUCTIONS DE PERIPHERIQUES

1. Le light-pen

• Principe de fonctionnement

Le crayon optique contient deux appareils totalement indépendants :

- un interrupteur, situé dans le nez du crayon ;
- un phototransistor, situé à l'intérieur du crayon.

Ce dernier, lorsqu'il reçoit suffisamment de lumière, laisse passer le courant électrique venant du MO5 ou, au contraire, bloque ce courant s'il est en face d'une zone sombre.

Chaque point de l'écran étant allumé (ou éteint) à un instant donné, tout l'écran est balayé point par point à une vitesse telle que, persistance rétinienne aidant, on a l'impression de les voir tous éclairés simultanément.

Placé face à un point (ou un paquet de points) de l'écran, le crayon optique indiquera donc si ce point est allumé ou éteint. Puisque c'est le MO5 lui-même qui gère l'écran, il connaît à chaque instant l'état et la position des points sur l'écran, donc la position du crayon optique.

Nous allons donc pouvoir disposer de fonctions Basic combinant les propriétés du light-pen et les possibilités graphiques de gestion de l'écran.

• Les instructions Basic

INPEN C,L

Cette instruction renvole Immédiatement les coordonnées du point graphique de l'écran placé face au crayon optique.

C est le numéro de colonne, compris entre Ø et 319.

L est le numéro de ligne compris entre Ø et 199.

Si le crayon optique ne détecte pas de lumière, il renvoie X = −1 et Y = −1.

INPUTPEN C,L

Cette instruction fonctionne comme la précédente, seulement dans le cas où l'interrupteur du crayon optique est enfoncé.

PTRIG

Cette instruction est une variable qui peut prendre la valeur −1 si l'interrupteur du crayon optique est actionné, ou la valeur 0 dans le cas contraire.

On l'utilisera le plus souvent dans des instructions de test du type : IF PTRIG = 0 THEN...
ou bien IF PTRIG = −1 THEN...

• Exemple d'application

Voici un petit programme qui vous permet de faire un dessin sur l'écran tant que vous appuyez le crayon sur celui-ci.

```
10 '          **************************
20 '          *                        *
30 '          * PROGRAMME DE DESSIN    *
40 '          *                        *
50 '          **************************
60 '
70 '
80 '
90 '
100 SCREEN4,6,6 : CLS
110 IF PTRIG =0 GOTO110
120 INPEN X,Y : IF X<0 OR Y<0 GOTO120 ELSE PSET(X,
Y)
130 IF PTRIG = 0 GOTO 110
140 INPEN X,Y : IF X<0 OR Y<0 GOTO 140 ELSE  LINE
-(X,Y)
150 GOTO130
160 END
```

TUNE

Cette instruction permet, par pointage du trait vertical qui apparaît sur l'écran, de régler le crayon optique.

2. Les manettes de jeux

Les manettes de jeux comprennent un bouton-poussoir et un levier. Le MO5 peut détecter :

- l'action sur le bouton,
- la position du levier par 1/8e de tour.

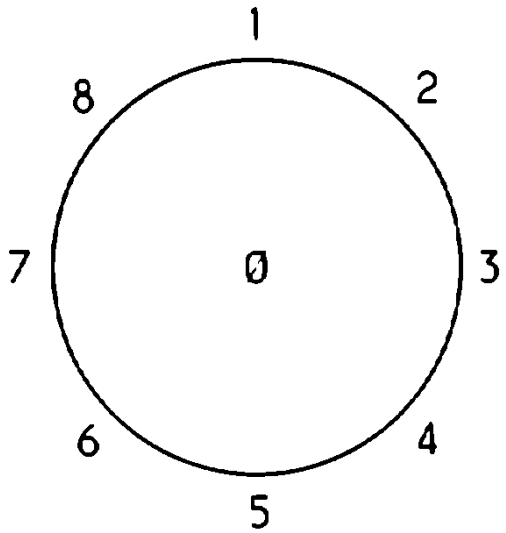


Fig. 11-1

STICK (N)

Cette instruction renvoie la position du levier de la manette numéro N.

Si le levier est vertical, STICK vaut Ø.
Si le levier est à midi, STICK vaut 1, etc.

Voir la figure XI-3 pour le codage complet.

STRIG (N)

Cette instruction, analogue à PTRIG, renvoie Ø si le bouton de la manette N n'est pas actionné, ou –1 si le bouton est actionné.

3. *La musique*

BEEP

Cette instruction déclenche une petite sonnerie, celle que l'on entend quand on tape sur le clavier, et que l'on peut obtenir également par CHR$(7).

PLAY A$

Cette instruction permet de jouer l'air de musique défini par la chaîne A$ de la façon suivante :

La chaîne contient la suite des notes que l'on veut jouer.

Chaque note est définie par deux lettres : DO – RE – MI – FA – SO – LA – SI et, éventuellement, les symboles dièse # ou bémol b.

Chaque note peut être choisie dans une octave parmi cinq. L'octave 1 (01) étant la plus grave et 05 la plus aiguë. Une octave couvre les notes de DO, RE... à SI, SI #.

La longueur de chaque note doit être précisée. Cette longueur ou durée peut varier de 1 à 96 par bond de 1. La noire a une longueur L24, donc la croche L12, la blanche L48...

On peut également définir l'amortissement de chaque note, entre Ø et 255.

Si l'amortissement est nul (AØ), le son est continu.

Le rythme de l'ensemble, ou tempo, peut varier de 1 à 255, 1 étant le rythme maximum correspondant donc à l'exécution la plus rapide.

On dispose en outre des silences P considérés comme des notes et suivant donc les règles ci-dessus.

Exemple : comment jouer « au clair de la lune » :

```
PLAY «DO DO DO RE L48 MI RE L24 DO MI RE RE L48 DO»
```

Au départ, par défaut, on considère :

- Attaque AØ
- Longueur L24
- Tempo T5
- Octave 04

On peut d'autre part manipuler la chaîne à jouer comme n'importe quelle variable chaîne.

4. *Les instructions de fichier*

• Description de fichier

Un fichier est un ensemble d'informations que l'on écrit ou qu'on lit sur un périphérique. Ce fichier possède en général un nom.

Suivant la nature des informations, on distingue plusieurs types de fichier :

- les fichiers de programmes,
- les fichiers de données,
- les fichiers binaires.

Pour accéder à un fichier, il faut toujours l'ouvrir. Puis il faut le fermer quand on a terminé de l'utiliser.

• Les types de périphériques

L'ordinateur MO5 possède les périphériques suivants :

- le lecteur-enregistreur de cassettes CASS
- l'écran SCRN
- l'imprimante parallèle LPRT
- le clavier KYBD

Lorsque l'on travaille avec l'imprimante parallèle, on peut définir un certain nombre d'options.

- Pour l'imprimante, on peut indiquer le nombre de caractères par ligne.

Les options seront donc représentées par un chiffre placé entre parenthèses à la suite du nom du périphérique, et en le séparant de ce dernier par : (deux points).

Suivra l'ensemble, le nom du fichier éventuellement. Le tout placé entre guillemets constitue le descripteur de fichier.

Exemple : « CASS:MATH »

Fichier sur cassette — Nom du fichier MATH

Le nom du fichier ne doit pas dépasser 8 caractères sans point ni virgule et peut être suivi d'un suffixe de 3 caractères, séparés des 8 premiers par un point.

Exemple : « LPRT : (8Ø) REPER.TEL »

Imprimante — Nbre de caract. par ligne — Nom — Suffixe

Lorsqu'on ne précise pas le périphérique choisi, c'est l'écran qui est retenu pour l'instruction LIST, et le lecteur-enregistreur pour les autres instructions.

• **Les instructions**

CLOSE # N°

Cette instruction ferme le fichier de numéro N. L'instruction CLOSE, utilisée seule, ferme tous les fichiers préalablement ouverts.

EOF (N°)

Cette instruction indique la fin d'un fichier. Elle prend la valeur Ø si la fin de fichier n'est pas atteinte, et la valeur –1 dans le cas contraire. Elle est en général utilisée dans des tests.

INPUT # N°, Var1, Var2

Cette instruction lit séquentiellement, sur le fichier de N° indiqué, les variables qui suivent comme pour une instruction INPUT.

Un exemple d'utilisation de cette instruction a été donné dans le cas du « Répertoire téléphonique ».

LINE INPUT # N°, Var$

Cette instruction permet de lire, sur le fichier de N° indiqué, la variable chaîne.

LIST

Cette instruction permet de lister sur l'écran la totalité du programme en mémoire si celui-ci n'est pas verrouillé.

On peut lister la partie du programme comprise entre les lignes NL1 et NL2 en ordonnant :

LIST NL1-NL2

Si on supprime NL1, on liste tout le programme jusqu'à la ligne NL2.
Si on supprime NL2, on liste tout le programme depuis NL1.

L'instruction LIST NL permet de ne lister que la ligne de numéro NL.

LIST « CASS : NOM », NL1-NL2

Cette instruction permet la sauvegarde sous forme d'un listing d'un programme appelé NOM, des lignes NL1 à NL2.

LIST «LPRT : (NC) », NL1-NL2

Cette instruction permet la sortie vers le périphérique imprimante parallèle des lignes NL1 à NL2 d'un programme, à raison de NC caractères par ligne. Par défaut, NC = 40.

LOAD « NOM », R

Cette instruction permet de charger en mémoire un programme sauvegardé sur la cassette avec le même NOM.

Si R est présent, il y a lancement automatique en mode RUN.

LOAD« seul charge le premier programme rencontré.

MOTOR ON (OFF)

Ces instructions provoquent le défilement (ou l'arrêt) de la bande du lecteur enregistreur de cassettes si celui-ci était en mode PLAY ou RECORD.

MERGE « NOM »

Cette instruction permet de charger en mémoire un programme sauvegardé sur cassette sous le même NOM sous forme ASCII, en le mélangeant au programme déjà en mémoire.

Si deux lignes ont le même numéro, c'est celle qui vient du lecteur qui remplace celle déjà en mémoire.

Exemple :

Programme en mémoire :	Programme fichier « MAO »
10 CLS : SCREEN 0,7,7	20 PRINT « NUMERO 2 »
20 PRINT « NUMERO 1 »	30 END

On fait MERGE « MAO » et on obtient en mémoire :

```
10 CLS : SCREEN 0,7,7
20 PRINT « NUMERO 2 »
30 END
```

L'option R (Run) est autorisée comme pour LOAD.

OPEN « O » / « I » , # N°, « PERIPH : (V), NOM »

Cette instruction ouvre le fichier numéro N°, sur le périphérique PERIPH, avec l'option V. Le fichier est nommé NOM.

Si OPEN « O » on établit la liaison MO5 → PERIPH

Si OPEN « I » PERIPH → MO5.

Le numéro du fichier doit être compris entre 1 et 16.

PRINT # N°, Var, Var

Cette instruction écrit sur le fichier N° les valeurs des variables indiquées en les séparant par un enregistrement du « Retour chariot .

PRINT # N° USING...

Cette instruction fonctionne comme PRINTUSING mais cette fois l'écriture se fait sur le fichier de numéro N°...

POS (# N°)

Cette instruction donne la position du curseur dans la ligne en cours du fichier N°...

SAVE « NOM »

Permet la sauvegarde du programme NOM sur le lecteur enregistreur.

Si on ajoute l'option **,A**, cette sauvegarde se fait sous forme ASCII de façon à pouvoir être utilisée par MERGE.

Si on ajoute l'option **,P**, cette sauvegarde se fait sous forme protégée, ce qui empêche le listage et la recopie à partir de la mémoire.

SKIPF « NOM »

Cette instruction permet de sauter le fichier NOM sans l'enregistrer et de se placer immédiatement après.

Elle est très utile pour positionner la bande avant un enregistrement.

• **Cas d'un fichier binaire**

Les informations binaires (programme en assembleur, mémoire d'écran, GR$...) peuvent être stockées sur cassette et relues grâce aux instructions suivantes :

LOADM « NOM », OFS, R

Cette instruction permet le chargement en mémoire du fichier binaire NOM. Si OFS est absent, le chargement est implanté aux adresses qui étaient siennes lors de la sauvergarde. Si OFS est présent, l'implantation est décalée d'un offset égal à la valeur de OFS.

R est l'option RUN.

SAVEM « NOM », Adr1, Adr2, Adr3

Cette instruction permet la sauvegarde sur cassette du contenu mémoire de Adr1 jusqu'à Adr2, sous forme d'un fichier binaire appelé « NOM ».

Adr3 est l'adresse d'exécution, dans le cas d'un programme en assembleur.

Adr3 = Adr1 dans le cas de sauvegarde des GR$.

Adr3 = &HØØØØ dans le cas de sauvegarde du contenu de la mémoire d'écran.

5. Programmes d'application

• **Création d'un fichier de données : le répertoire**

Dans le chapitre précédent, nous avions utilisé un fichier existant comprenant les noms, prénoms, adresses et téléphone de 5Ø personnes, pour faire une recherche d'adresse.

Ce fichier était ouvert en lecture avec le N° 2 :

1Ø OPEN « I », # 2, « REPER »

Puis les informations étaient rentrées quatre par quatre :

3Ø INPUT # 2, NOM$ (N), PRE$ (N), ADR$ (N), TEL$ (N)

Enfin le fichier 2 était fermé lignes 15Ø ou 16Ø par CLOSE # 2.

Pour créer ce fichier, on peut utiliser le programme suivant :

```
10 '         ****************************
20 '         *                          *
30 '         * CREATION DU REPERTOIRE   *
40 '         *                          *
50 '         ****************************
60 '
70 '
80 '---------OUVERTURE DU FICHIER-----
90 '
100 OPEN"O",#1,"CASS:REPER"
110 '
120 '
130 '--------LECTURE DES DONNEES------
140 '
150 '
160 FOR N=1 TO 50
170     INPUT"NOM : ",N$
180     INPUT"PRENOM : ",P$
190     INPUT"ADRESSE : ",A$
200     INPUT"TELEPHONE : ",T$
210 '
220 '
230 '--------ECRITURE SUR LE FICHIER---
240 '
250 '
260     PRINT#1,N$,P$,A$,T$
270     NEXTN
280 '
290 '
300 '--------FERMETURE DU FICHIER------
310 '
320 '
330 CLOSE#1
340 END
```

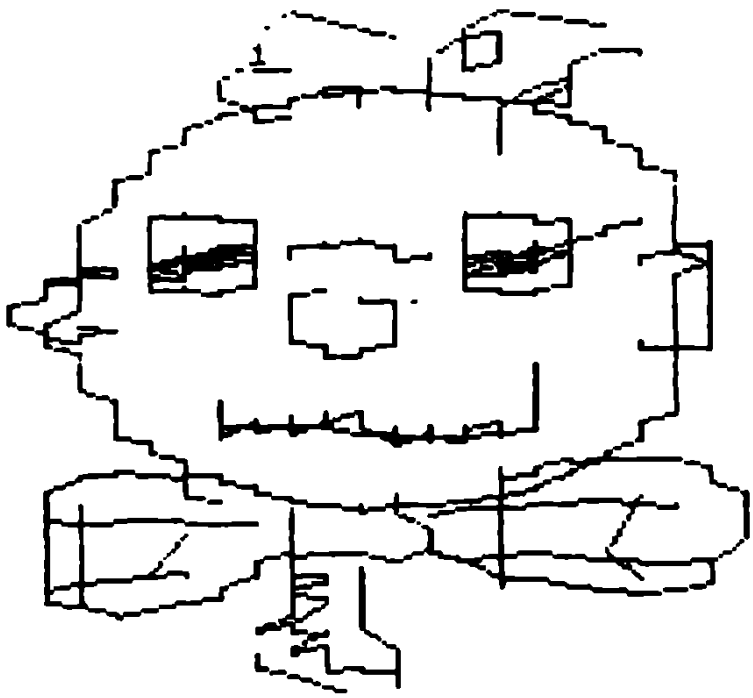
• Dessin sur l'écran et sauvegarde du dessin

On définit tout d'abord une zone de la dimension d'un caractère dans la case (X = 39, Y = 24) en bas et à droite de l'écran. Si le crayon optique pointe cette case, alors on sort du programme de dessin proprement dit pour sauver ce dessin sur cassette sous forme d'un fichier binaire.

On sauvegarde d'abord la forme du dessin, puis la couleur des points, ces informations étant stockées indépendamment en mémoire.

```
10 '          ************************
20 '          *                      *
30 '          * DESSIN SUR L'ECRAN   *
40 '          *                      *
50 '          ************************
60 '
70 '
80 '--------EFFACEMENT DE L'ECRAN ET PLACEMENT DE
LA CASE 39.24-----------------
90 '
100 CONSOLE 0,24 :SCREEN 0,7,7 : CLS
110 PSET (39,24)"."
120 '
130 '
140 '--------DEFINITION DE LA ZONE 0-----
150 '
160 '
170 PEN0;(39,24) : ONPEN GOTO 400
180 '
190 '
200 '--LECTURE ET PLACEMENT D'UN POINT--
210 '
220 '
230 INPUTPEN C1,L1
240 PSET(C1,L1)
250 IF PTRIG=0 GOTO 170
260 '
```

```
270 '
280 '----LECTURE D'UN POINT ET TRAIT----
290 '
300 '
310 INPUTPEN C2,L2
320 IF C2<0 OR L2<0 GOTO 240
330 LINE -(C2,L2)
340 IF PTRIG = 0 GOTO 170 ELSE GOTO 250
350 '
360 '
370 '----SAUVEGARDE DU DESSIN----------
380 '
390 '
400 POKE &HA7C0,PEEK(&HA7C0) OR 1
410 '
420 '
430 '-----En &HA7C0 se trouve le PORTA du 6821 sys
teme dont le bit PA0 permet de selectionner la mem
oire de forme de l'ecran (par mise a 1) ou la memo
ire de couleur (par mise a 0).-------------------
440 '
450 '
460 '
470 SAVEM "DESSIN",&H0000,&H1F40
480 POKE &HA7C0,PEEK(&HA7C0) AND 254
490 SAVEM"COULEUR",&H0000,&H1F40
```



La recopie du dessin de la cassette vers l'écran sera également faite en deux temps : tout d'abord la forme, puis la couleur.

```
10 '          ************************
20 '          *                      *
30 '          * RECOPIE D'UN DESSIN  *
40 '          *                      *
50 '          ************************
60 '
70 '
80 '----RECUPERATION DU DESSIN-----
90 '
100 POKE &HA7C0,PEEK(&HA7C0) OR 1
110 LOADM "CASS:DESSIN"
120 '
130 '
140 '----RECUPERATION DES COULEURS-
150 '
160 '
170 POKE &HA7C0,PEEK(&HA7C0) AND 254
180 LOADM"CASS:COULEUR"
190 END
```

• Création musicale

Ce programme réserve tout d'abord la place pour un GR$ et définit ce GR$, qui sera une note noire.

On peut rentrer par le clavier jusqu'à 20 notes.

Après avoir tracé une portée, on tape la note que l'on veut jouer : DO, RE, MI, FA, SO, LA, SI, PP.

On remarque que le SOL sera tapé SO et le silence PP.

Chaque note tapée est jouée et placée sur la portée. Lorsque 20 notes ont été tapées, toute la partition est rejouée.

```
1 '          **********************
2 '          *                    *
3 '          * CREATION MUSICALE  *
4 '          *                    *
5 '          **********************
6 '
7 '
8 '--INITIALISATION, DESSIN D'UNE NOTE--
9 '
10 CLEAR .,1 : CLS : DIM NO$(20)
20 DEFGR$(0) = 24,126,126,255,255,126,126,24
22 '
24 '
25 '------DESSIN DE LA PORTEE-----
26 '
28 '
30 FOR L=A TO 68 STEP 16
40     LINE (0,L) - (319,L)
50     NEXTL
60 X = 0 : Y = 10 : N = 1
70 IF N= 21 GOTO 120
72 '
74 '
75 '-----LECTURE DE LA NOTE------------
76 '
78 '
80 NO$(N) = INPUT$(2)
```

```
90 RESTORE 200
92 '
94 '
95 '-RECHERCHE DE LA NOTE DANS LA TABLE-
96 '
98 '
100 READ N$
110 IF NO$(N) = N$ THEN LOCATE X,Y : PRINTGR$(0) :
 PLAY"L96"+N$ : X = X+2 : Y=10 : N = N+1 : GOTO 70
 ELSE Y =Y-1 : GOTO 100
120 FOR N=1 TO 20
130     PLAY"L96"+NO$(N)
140     NEXTN
150 END
192 '
194 '
195 '------TABLE DES NOTES ----------
196 '
198 '
200 DATA DO,RE,MI,FA,SO,LA,SI,PP
```

Chapitre 12

LES AUTRES FONCTIONS

1. Ecriture et lecture en mémoire

Lorsqu'on le souhaite, on peut écrire des informations binaires, sous forme d'un octet, soit en mémoire vive (RAM), soit dans un registre de circuit d'interface pour périphérique.

Dans un cas comme dans l'autre, il faut connaître l'adresse de la mémoire ou du registre (Adr) pour pouvoir y écrire l'octet (Oct).

POKE Adr, Oct

Cette instruction permet de ranger l'octet Oct à l'adresse Adr.

On peut également lire le contenu d'une adresse de mémoire (morte ou vive) ou d'un registre.

PEEK (Adr)

Cette instruction prend la valeur (décimale) du contenu de l'adresse. Cette valeur peut être testée ou affectée à une variable.

Exemple d'utilisation

Comme pratiquement tous les microprocesseurs, le 6809 lors de son initialisation (Reset) va chercher l'adresse de départ du programme d'initialisation en &HFFFE, &HFFFF. Donc, pour le réinitialiser par programme, il faut faire :

```
AD = PEEK (&HFFFE) * 256 + PEEK (&HFFFF)
EXEC AD
```

VARPTR (Var)

Cette instruction renvoie l'adresse du premier octet de la variable indiquée. Donc :

– pour les variables numériques entières, cette instruction renvoie l'adresse de l'octet de poids fort. L'octet de poids faible se trouve en VARPTR (Var) + 1,

– pour les variables numériques de simple et double précision, cette instruction renvoie l'adresse de l'exposant, les octets suivants formant la mantisse,

– pour les variables chaîne, cette instruction renvoie l'adresse de l'octet contenant la longueur de la chaîne, les deux octets suivants contenant le poids fort et le poids faible de l'adresse du premier caractère de la variable.

Exemple :

10 A$ = « BONJOUR » : N = 1

L'adresse du 1er caractère « B » de A$ est :

AD = PEEK (VARPTR (A$) + 1) * 256 + PEEK (VARPTR (A$) + 2)

Donc l'instruction
PRINT PEEK (AD) renvoie B
PRINT PEEK (AD + 1) renvoie O, etc.

2. *Gestion des erreurs*

Lorsqu'une erreur est détectée dans un programme, celui-ci cesse le travail en cours et renvoie sur l'écran un message d'erreur, indiquant le type de l'erreur par un numéro de code, et le numéro de la ligne où s'est produite l'erreur.

Les instructions suivantes permettent de créer des sous-programmes de gestion des erreurs, évitant ainsi la procédure ci-dessus.

ERL

C'est une variable égale au N° de la ligne où s'est produite une erreur.

ERR

C'est une variable égale au numéro de code de l'erreur.

ERROR code

Cette instruction placée dans un programme agit exactement comme si l'erreur de numéro indiqué par code était détectée.

En temps normal, elle provoque donc l'apparition d'un message d'erreur sur l'écran.

Dans le cas où un sous-programme de gestion aura été implanté, cette instruction entraîne l'appel de ce sous-programme.

ON ERROR GOTO N°L

Cette instruction, placée en début de programme, indique qu'un sous-programme de gestion des erreurs a été implanté en N° L et que l'on doit s'y rendre dès qu'une erreur est détectée.

RESUME

Cette instruction doit obligatoirement être placée en fin du sous-programme de gestion des erreurs. Elle permet le retour à la ligne où s'est produite l'erreur.

RESUME NEXT

Permet le retour à la ligne qui suit celle où s'est produite l'erreur.

RESUME N° L

Permet le retour au numéro de ligne indiqué.

Exemple d'utilisation

Lors d'une utilisation de l'imprimante comme fichier de sortie, si l'opérateur oublie de mettre l'imprimante sous tension, le MO5 renvoie un message d'erreur. On y remédie comme suit :

```
1 '         ********************************
2 '         *                              *
3 '         * ECRITURE SUR L'IMPRIMANTE    *
4 '         *                              *
5 '         ********************************
6 '
7 '
8 '-----GESTION DE L'ERREUR------------
9 '
10 ON ERROR GOTO 100
12 '
14 '
15 '----OUVERTURE DU FICHIER-----------
16 '
18 '
20 OPEN "O",#1,"LPRT:(80)"
30 PRINT"Vous pouvez taper votre texte."
```

```
32 '
34 '
35 '--LECTURE/ECRITURE D'UN CARACTERE--
36 '
38 '
40 A$ = "" : A$ = INKEY$ : IF A$="" THEN GOTO 40
50 PRINT#1,A$; : PRINTA$;
60 IF A$ = CHR$(13) THEN PRINT#1,CHR$(13) : PRINT
CHR$(13)
70 GOTO 40
92 '
94 '
95 '-----GESTION DE L'ERREUR-----------
96 '
98 '
100 IF ERR = 53 THEN PRINT"Mettez l'imprimante sou
s tension."
110 PRINT"Est-ce fait ?"
120 X$ = INPUT$(1)
130 IF X$ = "O" THEN RESUME
140 IF X$ = "N" THEN END
200 PLAY"DOMIDOPPPDOMIDO"
210 GOTO 120
```

3. Les commandes Basic

CONT

Cette instruction permet de redémarrer un programme là où on l'avait stoppé par action sur CNT C ou lors de l'exécution d'un STOP.

DELETE NL1-NL2

Cette instruction efface du programme toutes les lignes comprises entre NL1 et NL2 incluses.

GOTO NL

Cette instruction permet le lancement d'un programme à la ligne NL sans réinitialiser les variables.

LIST

Cette instruction permet de lister un programme sur l'écran. On ne peut visualiser :
- qu'une ligne par LIST N° L,
- que jusqu'à une ligne par LIST – N° L,
- que depuis une ligne par LIST N° L –,
- qu'entre deux lignes par LIST N° L1 – N° L2.

Pour lister sur l'imprimante, il faut donner le nom de ce périphérique et, en option, le nombre de caractères par ligne.

Exemple :

LIST « LPRT : (80) », 10 – 100

NEW

Cette instruction efface le programme en mémoire.

RUN

Cette instruction lance un programme à la première ligne en réinitialisant toutes les variables. On peut éventuellement donner le numéro de ligne de départ.

Exemple : RUN 100

RUN peut être également utilisé en remplacement de LOAD « NOM »,R. Dans ce cas, il charge le fichier présent sur le périphérique désigné, puis le lance comme un RUN de commande directe.

L'option, R (exemple : RUN « TOTO », R) permet le maintien des fichiers à l'état ouvert.

SAVE, LOAD

Voir « Les instructions de périphériques ».

TRON

Cette instruction, utile pour la mise au point des programmes, permet l'affichage sur l'écran des numéros des lignes exécutées pendant qu'un programme tourne.

TROFF

Cette instruction supprime l'action de TRON.

3 Dessins et graphiques

Chapitre 13

LES ZONES GRAPHIQUES EN RAM

Comme nous l'avons vu dans les chapitres consacrés à la gestion de l'écran et aux caractères utilisateur, l'écran est divisé en 40 colonnes et 25 lignes, soit 1 000 cases « caractères ». D'autre part, un caractère est une matrice de 8 × 8 = 64 points.

Chaque point représente la forme ou le fond d'un caractère ou d'un dessin et les points sont regroupés sur chaque ligne par paquets de huit, appelés GROUPE DE POINTS LIGNE (GPL).

Exemple : pour X = Ø à X = 7 se trouve le 1er GPL, de X = 8 à X = 15 le second...

Dans un groupe de points ligne, chaque point fait partie, ou de la forme, ou du fond, et l'on ne pourra donc définir dans un GPL que deux couleurs, la couleur de forme et la couleur de fond.

La fenêtre écran dans laquelle nous travaillons est donc constituée de 40 × 200 = 8000 GPL, chacun d'eux pouvant être numéroté. Le GPL situé en haut et à gauche (Y = 0, X = 0 à 7) sera le GPL N° 0 ; puis, pour X = 8 à 15, le GPL 1, ..., pour Y = 1 et X = 0 à 7 nous trouverons le GPL N° 40,... et ainsi de suite jusqu'au GPL n° 7999 en bas et à droite de la fenêtre de travail.

Pour définir complètement un GPL, il faut tout d'abord indiquer si un point appartient à la forme ou au fond du dessin ; comme un GPL contient 8 points, il faut donc un octet pour le représenter (fig. 13-1).

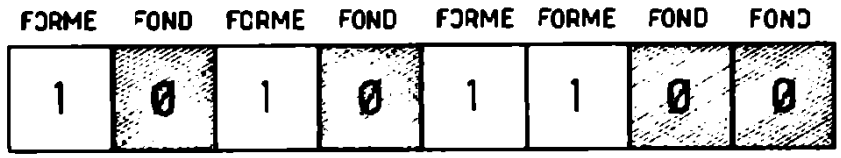


Fig. 13-1. – Représentation binaire d'un GPL.

- Lorsqu'un bit de cet octet est à 0, il appartient au fond.
- Lorsqu'un bit de cet octet est à 1, il appartient à la forme.

Pour représenter la totalité de la fenêtre, il faudra donc 8 K-octets de RAM.

Pour obtenir la couleur d'un point, on utilise les trois couleurs de base : bleu, vert et rouge. Pour définir une couleur, il faudra donc quatre bits représentant :

- l'absence de la couleur de référence si un bit est à zéro,
- la présence de la couleur de référence si ce bit est à 1,
- le mode saturé ou demi-teinte.

On obtient alors les seize couleurs du tableau XIII-1.

Pour chaque GPL, on peut définir la couleur des points appartenant à la forme, et la couleur de ceux appartenant au fond. On peut donc définir deux couleurs par GPL, ce qui nécessitera huit bits. Il faudra donc une mémoire couleur de 8 K × 8 bits.

En mode graphique les couleurs de fond doivent être complémentaires, c'est-à-dire :

NOIR	= -1	BLEU	= -5	GRIS	= - 9	BLEU CLAIR	= - 13
ROUGE	= -2	MAGENTA	= -6	ROSE	= - 10	ROSE PARME	= - 14
VERT	= -3	CYAN	= -7	VERT CLAIR	= - 11	CYAN CLAIR	= - 15
JAUNE	= -4	BLANC	= -8	JAUNE PALE	= - 12	ORANGE	= - 16

Bleu	Vert	Rouge	Saturé	Pastel
Ø	Ø	Ø	Ø Noir	8 Gris
Ø	Ø	1	1 Rouge	9 Rose
Ø	1	Ø	2 Vert	10 Vert clair
Ø	1	1	3 Jaune	11 Jaune clair
1	Ø	Ø	4 Bleu	12 Bleu clair
1	Ø	1	5 Magenta	13 Rose parme
1	1	Ø	6 Cyan	14 Bleu ciel
1	1	1	7 Blanc	15 Orange

Tableau 13-1. – Synthèse des teintes.

Chaque GPL a donc besoin de deux fois 8 K RAM pour définir les points et leur couleur. Deux blocs mémoire RAM sont placés aux mêmes adresses, chaque adresse correspondant au numéro d'ordre du GPL, un groupe mémorisant les points et l'autre les couleurs (voir fig. 13-2).

La sélection d'un bloc mémoire parmi les deux se fait par la mise à Ø ou à 1 du bit AØ du PORT A du circuit 6821, se trouvant à l'adresse &HA7CØ.

AØ à Ø sélectionne la mémoire de « couleur ».
AØ à 1 sélectionne la mémoire de « forme ».
(Voir le programme « dessin sur l'écran ».)

La mémoire écran se trouve donc de l'adresse &HØØØØ à l'adresse &H1F3F.

La couleur du cadre est définie par les bits PA1, PA2, PA3, PA4 du port A du 6821 (&HA7CØ).

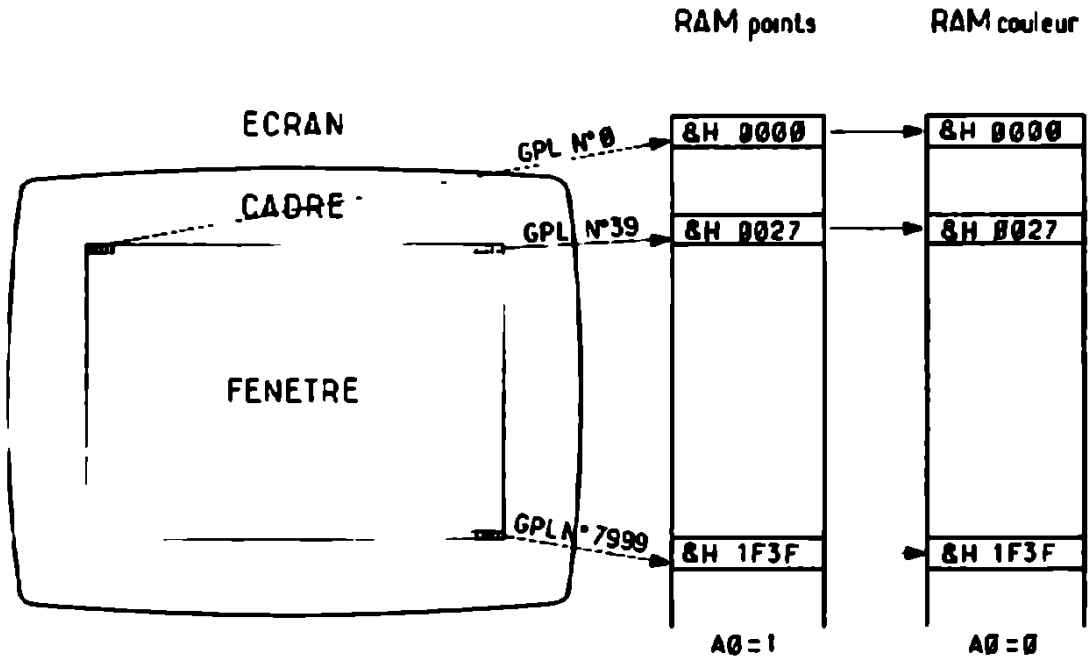


Fig. 13-2. – Mémoire d'écran.

Chapitre 14

LES FONCTIONS GRAPHIQUES

BOX (X_1, Y_1) – (X_2, Y_2) « Car », Attribut

Cette instruction permet le tracé des contours d'une boîte dont les coordonnées des sommets X_1, Y_1 et X_2, Y_2 correspondent à une diagonale.

Le contour de cette boîte peut être tracé à l'aide d'un caractère placé entre « ». Dans ce cas, l'attribut définit en trois valeurs (voir COLOR), la couleur des points, la couleur du fond, et l'inversion vidéo (couleurs de Ø à 15).

Le contour de la boîte peut être tracé en mode graphique. Dans ce cas, il n'y a ni caractère ni guillemets, et l'attribut ne définit que la couleur de ce contour-points à l'aide d'une valeur :

- de – 16 à – 1 si c'est un contour fond,
- de Ø à 15 si c'est un contour forme.

BOXF (X_1, Y_1) – (X_2, Y_2) « Car », Attribut

Cette instruction suit les mêmes règles que la précédente, mais dessine une boîte pleine.

LINE (X_1, Y_1) – (X_2, Y_2) « Car », Attribut

Cette instruction suit les mêmes règles que la précédente, mais tire une droite entre (X_1, Y_1) et (X_2, Y_2).

Si, dans ces trois instructions, (X_1, Y_1) est omis, le tracé se fait depuis le point courant, c'est-à-dire depuis (X_2, Y_2) de la dernière instruction de tracé utilisée.

PSET (X, Y) « Car », Attribut

Cette instruction dessine sur l'écran un point ou un caractère dont les couleurs précisées dans Attribut suivent les mêmes règles que précédemment.

POINT (X, Y)

Cette instruction renvoie la couleur (de – 16 à + 15) du point de coordonnées X, Y.

COLOR P, F, V

Cette instruction définit les couleurs points P et fond F ainsi que l'inversion vidéo si V est à 1.

SCREEN P, F, C, V

Comme la précédente, cette instruction définit les points P et fond F, l'inversion vidéo V, mais aussi la couleur du cadre C (de Ø à 15).

Chapitre 15

APPLICATIONS

Tracé de la courbe de réponse en fréquence d'un filtre R-C

Nous avons vu, au chapitre concernant les variables, que la fréquence de coupure du filtre R-C était $1/2\ \pi\ R_2 \cdot C$ et le gain en dB : $G = -4{,}34 * LN[\ 1 + (R_2 * C * 2 * \pi * f \uparrow 2]$.

Dans une première partie du programme, on calcule la valeur de f et on affiche le tableau des résultats G (dB) = f pour 15 valeurs de fréquences.

Essayons maintenant de tracer la courbe du gain en fonction de f en coordonnées semi-logarithmiques de 10 Hz à 500 kHz.

Nous disposons de 320 points horizontaux ; or la longueur d'une décade 10-100, 100-1 000 est constante et, de 100 kHz à 500 kHz, on occupe $\log_{10} 5 = 0{,}7$ fois cette longueur. La longueur en points écran d'une décade est donc 320/4,7 = 85 points maximum. On choisira une longueur de 64 points.

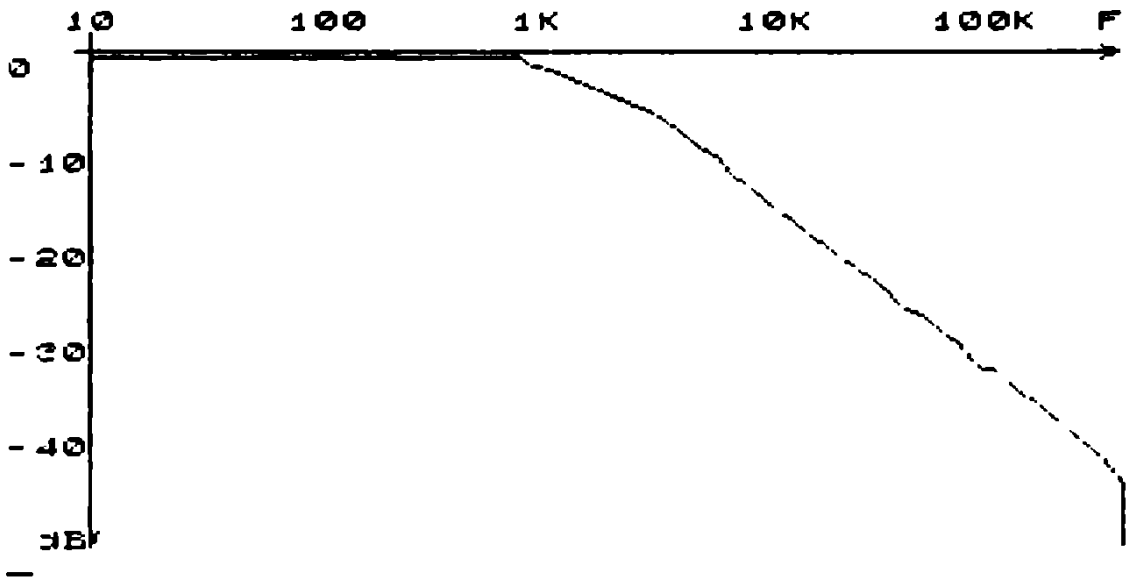
On dispose de 200 points en ordonnée. En prenant 1 dB = 3 points et en traçant la courbe jusqu'à – 40 dB, il suffit de 120 points, ce qui laisse suffisamment de place pour le titre, les axes et les valeurs du gain.

On en déduit donc le programme suivant.

```
10 '          ************************
20 '          *                      *
30 '          * REPONSE D'UN FILTRE  *
40 '          *                      *
50 '          ************************
60 '
70 '
80 '----------CALCUL DU GAIN------
90 '
```

```
95 CONSOLE 0,24 : SCREEN 2,0,0 :CLS
100 A$ = "###.##^" : B$ = "##" : C$ = "##^"
110 DIM G(15),F(15) : E = 4.3429419 : P = 3.1415927 : R2
 = 1 E 5
120 INPUT "Donner la valeur de C : ",C
130 F0 = 1/(2*P*R2*C) : CLS : PRINT"F0 = ";INT(F0);"Hz"
: PRINT
140 FOR I=1 TO 15
150     READ F(I)
160     G(I) = -E*LOG(1+(R2*C*2*P*F(I))^2)
170     PRINT"F("; : PRINTUSING B$;I; : PRINT") ="; : PR
INTUSING C$;F(I); : PRINT"    G("; : PRINTUSING B$;I; :
PRINT") ="; : PRINTUSING A$;G(I)
180     NEXT I
181 DATA 10,20,50,100,200,500,1E3,2E3,5E3,1E4,2E4,5E4,1E
5,2E5,5E5
182 '
184 '
185 '------------TRACE OU FIN---------
186 '
188 '
190 PRINT"Voulez-vous le trace de la courbe ?"
200 R$ =INKEY$ : IF R$="" GOTO 200 ELSE IF R$="N" THEN E
ND ELSE IF R$="O" GOTO 210 ELSE GOTO 200
202 '
204 '
205 '---------TRACE DES AXES----------
206 '
208 '
210 CLS : SCREEN 2,0,0
220 LINE (24,24)-(24,198) : LINE (22,193)-(24,198) : LIN
E (26,193)-(24,198)
230 LINE (20,30)-(318,30) : LINE (313,28)-(318,30) : LIN
E (313,32)-(318,30)
240 LOCATE 2,2 : PRINT"10"SPC(6)"100"SPC(5)"1K"SPC(6)"10
K"SPC(4)"100K"SPC(2)"F"
250 LOCATE 0,4 : PRINT"0" : LOCATE 0,8 : PRINT"-10" : LO
CATE 0,12 : PRINT"-20" : LOCATE 0,16 : PRINT"-30" : LOCA
```

```
TE 0,20 : PRINT"-40" :LOCATE1,24 : PRINT"dB"
262 '
264 '
265 '--------TRACE DE LA COURBE --------
266 '
268 '
270 PSET(24,24) : COLOR 1 : A =1
280 FOR N=10 TO 90 STEP 10
290       F = N*A : X= INT(28*LOG(F)-40)
300       G = INT(E*LOG(1+(R2*C*2*P*F)^2))
310       Y = 24+3*G
320       IF Y>=198 THEN END
330       LINE -(X,Y)
340       NEXT N
350 A = 10*A : IF A>100000 THEN END ELSE GOTO 280
```



4 Le micro-processeur 68Ø9

Chapitre 16

CARACTERISTIQUES GENERALES

Le 68Ø9 est un microprocesseur 8 bits, c'est-à-dire qu'il est capable de traiter les informations (les données) 8 par 8 (octet par octet). L'ensemble de ces huit informations binaires circulera dans tout le micro-ordinateur, le long de ce qu'on appellera le BUS de DONNEES.

La conception tant matérielle que logicielle du 68Ø9 fait de lui le microprocesseur idéal pour les techniques de programmation moderne (banalisation de l'instruction, réentrance des sous-pro-

grammes, programmation modulaire) et l'implantation de langages évolués de haut-niveau.

Bien que possédant moins d'instructions que le 6800 (59 au lieu de 72), l'augmentation très importante de ses modes d'adressage et du nombre de registres, ainsi que la possibilité d'appliquer pratiquement toute instruction à n'importe quel registre en font un des microprocesseurs les plus puissants actuellement sur le marché, puisqu'il permet 1464 instructions différentes (voir annexe 10).

Pour l'utilisateur du MO5, c'est principalement cette étude du logiciel qui est intéressante, car l'implantation matérielle est déjà réalisée, et seule nous intéresse la possibilité de programmer en assembleur.

Chapitre 17

STRUCTURE INTERNE

VOIR ANNEXE 1

1. Les accumulateurs A, B et D

Le 68Ø9 contient deux registres de travail de 8 bits appelés Accumulateurs A et B. Ces registres peuvent stocker chacun un octet et de nombreuses instructions leur sont applicables.

Par l'intermédiaire du bus interne du 68Ø9, ils sont donc reliés au bus de données. Ce dernier étant bidirectionnel, on pourra donc :

- faire entrer un octet dans un accumulateur,
- faire sortir un octet d'un accumulateur,
- manipuler un octet d'accumulateur, tout en restant à l'intérieur du 68Ø9.

Une particularité intéressante du 68Ø9 est la possibilité logicielle de considérer que ces deux accumulateurs n'en font plus qu'un, appelé accumulateur D, formé donc de 16 bits (2 octets) et obtenu en juxtaposant A et B, de telle façon que A contienne l'octet de poids fort et B l'octet de poids faible de D (voir fig. 17-1).

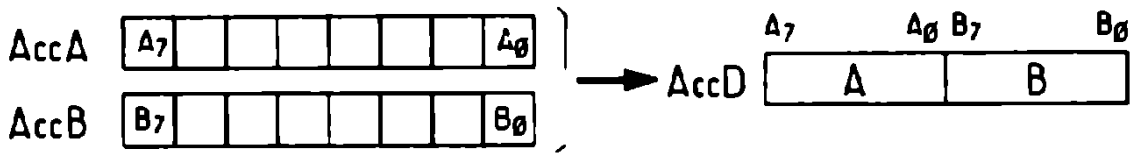


Fig. 17-1. – L'accumulateur D.

Des instructions existeront donc avec le 68Ø9 qui permettront de manipuler les données par paquets de 16 bits bien que le bus de données soit un bus 8 bits. C'est cette caractéristique qui fait souvent dire que le *68Ø9 est un microprocesseur pseudo 16 bits.*

2. Les registres S, U, X et Y

Le microprocesseur 68ØØ contenait un registre d'index X et un pointeur de pile S. Le 68Ø9 a fait plus que doubler ces caractéristiques. En effet, on peut y trouver deux registres d'index X et Y et deux pointeurs de pile S et U mais, en outre, la totale symétrie des instructions fait que les premiers peuvent jouer le rôle des seconds et vice-versa et que l'on peut échanger et transférer les contenus de tous ces registres entre eux, puisque ce sont tous en effet des registres 16 bits.

Etant reliés au bus de données on peut donc :

– faire entrer deux octets dans un registre, en deux fois,
– faire sortir deux octets d'un registre soit vers le bus de données, en deux fois, soit d'un seul coup vers le bus d'adresses,
– manipuler le contenu d'un registre de la façon interne au 68Ø9.

En particulier, grâce à la possibilité de concaténer A et B, on pourra échanger les contenus de tous les registres 16 bits entre eux, y compris l'accumulateur D et *même le compteur programme PC.*

3. Le registre compteur programme

Ce compteur 16 bits a pour rôle de faire passer sur le bus d'adresse l'adresse de la mémoire, du circuit d'interface,... auquel correspond le traitement de l'instruction en cours.

Grâce aux énormes possibilités logicielles du 68Ø9, on pourra considérer ce compteur comme un registre 16 bits ordinaire dont on pourra par exemple échanger le contenu avec D, X, Y, S ou U.

4. Le registre de page DP

Ce registre 8 bits est en permanence relié à la moitié haute du bus d'adresse. Il contient donc l'octet de poids fort de l'adresse, ce qui va permettre en mode d'adressage direct d'atteindre l'une quelconque des 65 535 adresses du MO5.

Alors qu'avec le 68ØØ on ne pouvait travailler en mode direct que sur la page Ø, c'est-à-dire des adresses &HØØØØ à &HØØFF, on peut cette fois aller de &HØØØØ à &HFFFF en modifiant le contenu de DP.

1re remarque : une fois DP chargé par un octet, le mode d'adressage direct ne s'étend que sur 256 adresses de (DP)ØØ à (DP)FF.

2e remarque : pour conserver sa compatibilité ascendante de logiciel avec le 6800, le 6809 remet à zéro le registre DP à la mise sous tension.

3e remarque : il n'y a pas d'instruction qui permette de charger directement ce registre DP avec l'octet choisi. Il faut donc charger, par exemple, un accumulateur puis échanger son contenu avec DP.

5. Le registre de codes condition CCR

Ce registre 8 bits donne des indications sur le résultat de la dernière opération réalisée par le 6809 et permet également de masquer des interruptions ou de connaître l'état du 6809 lors d'une opération d'empilage ou de dépilage.

Chaque bit de ce registre est indépendant des autres et contient une information à lui seul (voir fig. 17-2).

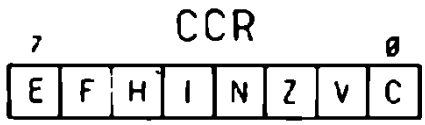


Fig. 17-2. – Le registre de codes condition.

Le BIT 0 (C)

Ce bit indique, quand il passe à 1, que la dernière opération a donné lieu à une retenue.

Lors d'une soustraction (en complément à 2), il représente donc le complément de la retenue.

Le BIT 1 (V)

Ce bit Indique, quand il passe à 1, que la dernière opération (en complément à deux signé) a donné lieu à un débordement, c'est-à-dire que la retenue du bit de poids le plus fort n'est pas égale à celle du bit immédiatement inférieur.

Exemple :

On veut additionner les valeurs décimales – 63 et – 110.

+ 63 s'écrit en binaire : 0 0 1 1 1 1 1 1

Pour obtenir – 63, on complémente à 1 puis on ajoute 1 (complément à 2) :

$$\begin{array}{r} +\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0 \\ 1 \\ \hline 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \end{array}$$

Le bit de poids le plus fort représente le signe ,0 pour + et 1 pour – et permet donc de savoir si les bits qui le précèdent représentent la valeur immédiate du nombre où son complément à deux.

De la même façon :

	+	1	1	0	s'écrit	0	1	1	0	1	1	1	0
et	–	1	1	0		1	0	0	1	0	0	1	0

Pour reconvertir un nombre binaire en complément à deux signé, on ne tient compte que des 7 bits de poids faibles dont on prend la valeur décimale (exemple ci-dessus, 18) et l'on retranche cette valeur de 128 (exemple ci-dessus, 110), puis l'on replace le signe moins.

Additionnons – 63 et – 110 :

$$\begin{array}{rr} & 0 \\ & 1\ 1\ 0\ 0\ 0\ 0\ 0\ 1 \\ & +\ 1\ 0\ 0\ 1\ 0\ 0\ 1\ 0 \\ \hline \text{Retenue C} \rightarrow & \boxed{1}\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \end{array}$$

La retenue entre les bits 6 et 7 est nulle, alors que celle portant sur les bits 7 est égale à 1. Le bit de retenue C passe donc à 1 ainsi que le bit de débordement V. On en déduit donc que le résultat est négatif et dépasse la valeur maximum représentable sur 8 bits, soit – 128.

Le BIT 2 (Z)

Ce bit indique, quand il passe à 1, que le résultat de la dernière instruction est nul.

Le BIT 3 (N)

Ce bit indique, quand il passe à 1, que le résultat de la dernière instruction est négatif.

Le BIT 4 (I)

Ce bit, lorsqu'il est à 1, empêche la prise en compte des interruptions IRQ.

Le BIT 5 (H)

Ce bit indique, quand il passe à 1, que la dernière opération a donné lieu à une retenue des bits 3 sur les bits 4, c'est-à-dire une retenue du premier demi-octet sur le deuxième (poids forts).

On l'appelle bit de demi-retenue, et il est surtout utilisé dans les opérations portant sur les valeurs décimales.

Le BIT 6 (F)

Ce bit, lorsqu'il est à 1, masque les interruptions du type FIRQ en n'autorisant que les interruptions plus prioritaires.

Le BIT 7 (E)

Ce bit indique, quand il est à 1, que tous les registres du 6809 ont été sauvegardés dans la pile.

Tous ces bits permettront de faire des tests, suite à une opération, afin d'orienter convenablement la suite du programme vers l'une ou l'autre partie suivant qu'un bit sera à 0 ou à 1.

6. Notion de pile

Lorsque l'on veut quitter un programme pour aller vers un sous-programme, il est nécessaire de sauvegarder en mémoire l'état des divers registres du microprocesseur, avant le départ en sous-programme, afin qu'au retour on puisse récupérer toutes ces valeurs.

Les cases mémoires sont toutes adjacentes, ce qui donne l'impression d'une pile, un peu comme une pile d'assiettes qu'on remplirait par le bas (adresse la plus forte) en remontant vers le haut (adresses de plus en plus faibles).

L'ordre d'empilement est toujours le même : on commence par ranger PC au fond de la pile, puis U juste au-dessus, puis dans l'ordre Y, X, DP, B, A et CCR (voir fig. 17-3). Dans cet exemple, on travaille avec la pile S, et à chaque instant le registre 16 bits S, appelé POINTEUR de PILE, contient l'adresse de la dernière case mémoire utilisée. Il pointe donc le haut de la pile.

Le dépilement se fait exactement dans l'ordre inverse car, comme pour une pile d'assiettes, on prend d'abord celle du dessus, puis celle immédiatement en dessous, etc.

La mémoire étant une mémoire 8 bits, on ne peut y ranger un registre 16 bits d'un seul bloc. On commence donc toujours par l'octet de polds faible (LOW) exemple PCL, puis par celui de poids fort (HIGH), exemple PCH. C'est ce qui explique qu'une sauvegarde complète des registres du 68Ø9 nécessite 12 cases mémoires.

Un empilage complet fait passer le BIT E à 1.

La pile S est en général la pile système, alors que la pile U est réservée à l'utilisateur. N'oublions pas que ces deux registres peuvent servir également pour l'adressage indexé.

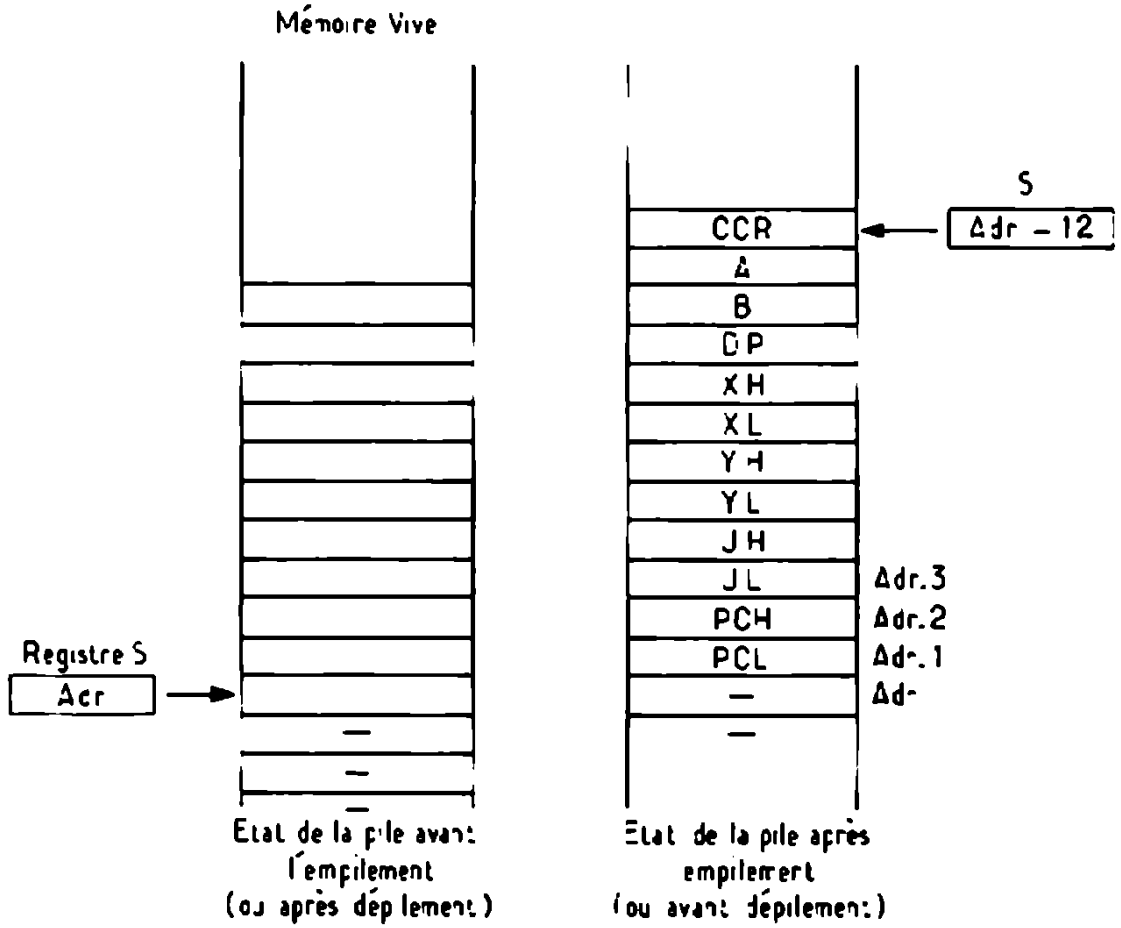


Fig. 17-3. – Manipulation de la pile.

Chapitre 18

LES MODES D'ADRESSAGE

1. Adressage inhérent

Dans ce mode d'adressage, il n'y a aucun échange avec l'extérieur du microprocesseur. On l'appelle également adressage Implicite, en ce sens que justement il n'y a pas à fournir d'adresse de départ ou de destination des données. Les instructions de ce type n'utilisent qu'un ou deux octets comme code machine hexadécimal. Elles sont les plus rapides à exécuter.

Exemple : l'instruction COMA, qui prend le contenu de l'accumulateur A et le complémente à 1 pour le remettre dans A, a pour code machine &H43 et nécessite deux cycles pour son exécution.

Cette opération est décrite comme suit : $\overline{A} \rightarrow A$.

A la suite de celle-ci, les bits N et Z prennent la valeur correspondant à A :

- si A = 00000000 Z = 1 sinon Z = 0
- si le bit de poids fort de A est 1 → N = 1 sinon N = 0

Le bit V est forcé à 0
Le bit C est forcé à 1
Le bit H est inchangé.

On vérifiera ce fonctionnement en se reportant à l'annexe « Jeu d'instructions du 6809 ».

Exemple : L'instruction EXG R1, R2 qui permet l'échange des contenus des registres R1 et R2 a un code machine sur deux octets et nécessite huit cycles machine.

Le premier octet, qui indique que l'opération est un échange, a pour code machine &H1E.

Le deuxième octet contient les codes du registre SOURCE et du registre DESTINATION (appelés comme ceci pour l'opération de transfert TFR, car pour EXG on met R1 dans R2 et R2 dans R1).

Les codes des registres sont les suivants :

0000 = D (A:B)	1000 = A
0001 = X	1001 = B
0010 = Y	1010 = CCR
0011 = U	1011 = DP
0100 = S	
0101 = PC	

Si l'on veut échanger les contenus de X et Y, on peut écrire :

1E 12, c'est-à-dire EXG X,Y

ou : 1E 21, c'est-à-dire EXG Y,X

En effet le code du registre source occupe le 1/2 octet de poids fort et le code du registre destination le 1/2 octet de poids faible (voir fig. 18-1).

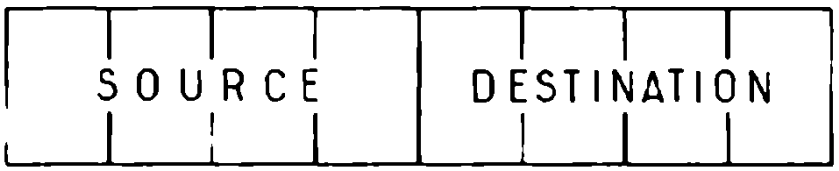


Fig. 18-1. – POST-BYTE des instructions TFR/EXG.

Cet octet supplémentaire est appelé POST-BYTE. On remarquera que, bien évidemment, si on peut échanger entre eux deux registres 16 bits ou deux registres 8 bits, il est par contre impossible d'échanger un registre 16 bits avec un registre 8 bits.

2. *Adressage immédiat*

Dans ce mode d'adressage, la valeur à traiter suit immédiatement le code opération.

Donc, si l'opération concerne un registre 8 bits, il faudra deux octets de code.

Si l'opération concerne un registre 16 bits, il faudra trois octets de code.

Ce mode d'adressage reste encore parmi les plus rapides (de 2 à 4 cycles).

Exemples : On peut charger un accumulateur, A par exemple, avec la valeur numérique &H1A en écrivant :

LDA # $ 1A	86 1A
Mnémonique assembleur	Code hexadécimal

En langage d'assemblage, le # symbolise l'adressage immédiat, le $ symbolisant toujours une valeur hexadécimale.

– On peut ajouter une valeur à l'accumulateur D :
ADDD # $ 1001 → C3 1001

(Voir fig. 18-2)

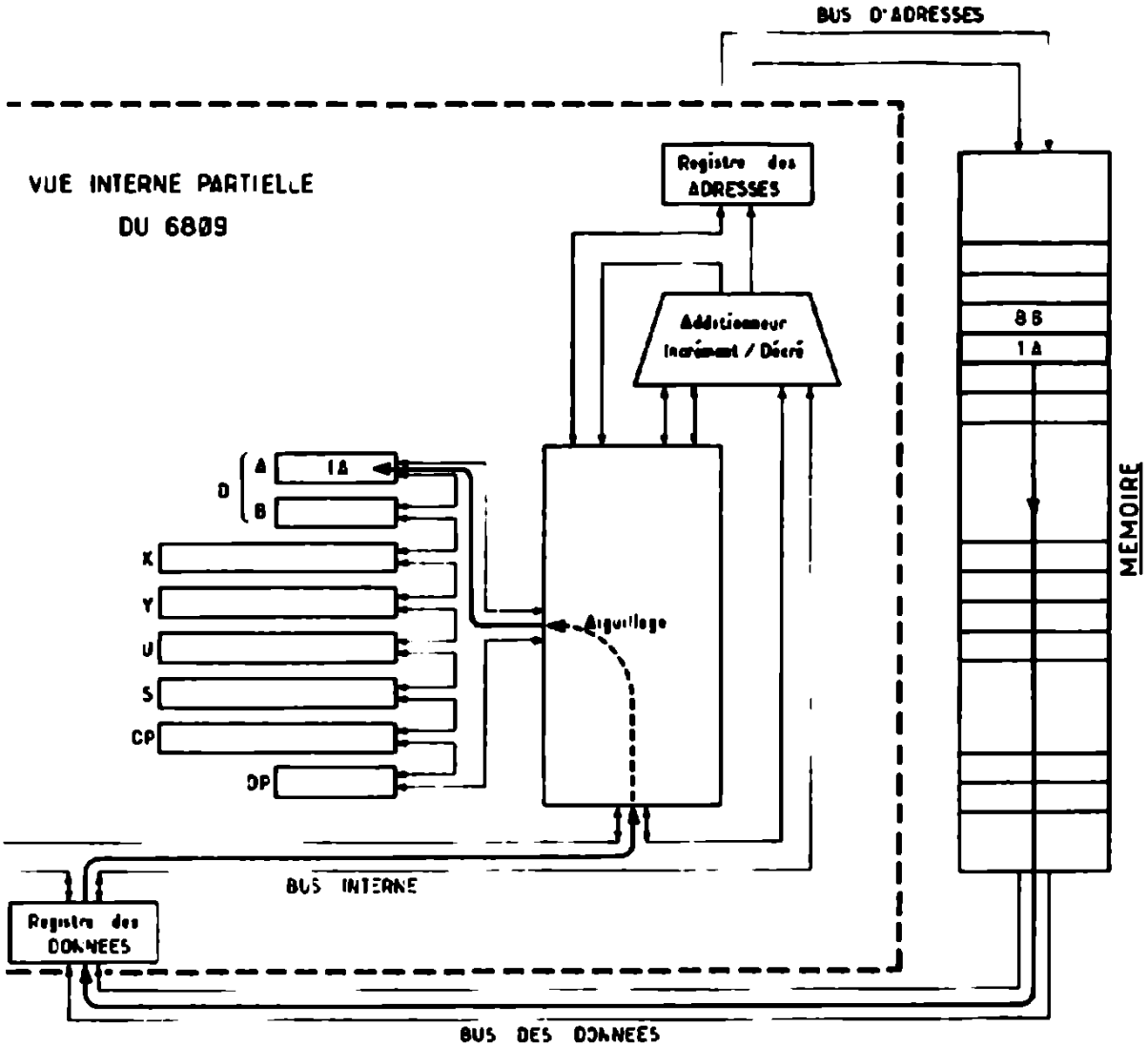


Fig. 18-2. – Adressage immédiat.

– On peut comparer un registre d'Index à une valeur numérique :
CMPX # $ 1EF3 → 8C 1E F3

3. Adressage étendu

Dans ce mode d'adressage, la valeur numérique à traiter se trouve rangée à une *adresse* connue, et c'est cette adresse que l'on indique *sur deux octets* à la suite du code opération.

Dans presque tous les cas, il faudra donc trois octets pour définir toute l'instruction. Seules exceptions, les quelques instructions dont le code opération tient sur deux octets.

La durée de traitement est de cinq à huit cycles.

Exemples :

– On veut charger l'accumulateur B avec une donnée qui se trouve à l'adresse $EFØØ :

LDB $ EFØØ → F6 EF ØØ

On remarquera la disparition du # de la syntaxe assembleur.

– On veut charger le pointeur U avec une donnée qui se trouve à l'adresse $8Ø1C. En fait, le pointeur U étant un registre 16 bits ne peut être chargé que par des informations de 16 bits, ce qui nécessite deux adresses mémoire consécutives, c'est-à-dire, dans l'exemple présent, $8Ø1C et $8Ø1D. Mais on n'indique dans la syntaxe que l'adresse la plus basse (qui contient l'octet de poids fort de la donnée, alors que Adresse + 1 contient l'octet de poids faible).

LDU $ 8Ø1C → FE 8Ø 1C

(Voir fig. 18-3)

– Pour charger le pointeur S avec la même donnée, on écrira :

LDS $ 8Ø1C → 1Ø CE 8Ø 1C

Code op.

Mais le code opération de cette instruction occupe deux octets.

4. Adressage direct

Dans ce mode d'adressage, la valeur numérique à traiter se trouve à une adresse connue et on indique, derrière le code opération, l'octet de poids faible de cette adresse. Ceci suppose que le registre de page ait été préalablement chargé avec l'octet de poids fort de l'adresse.

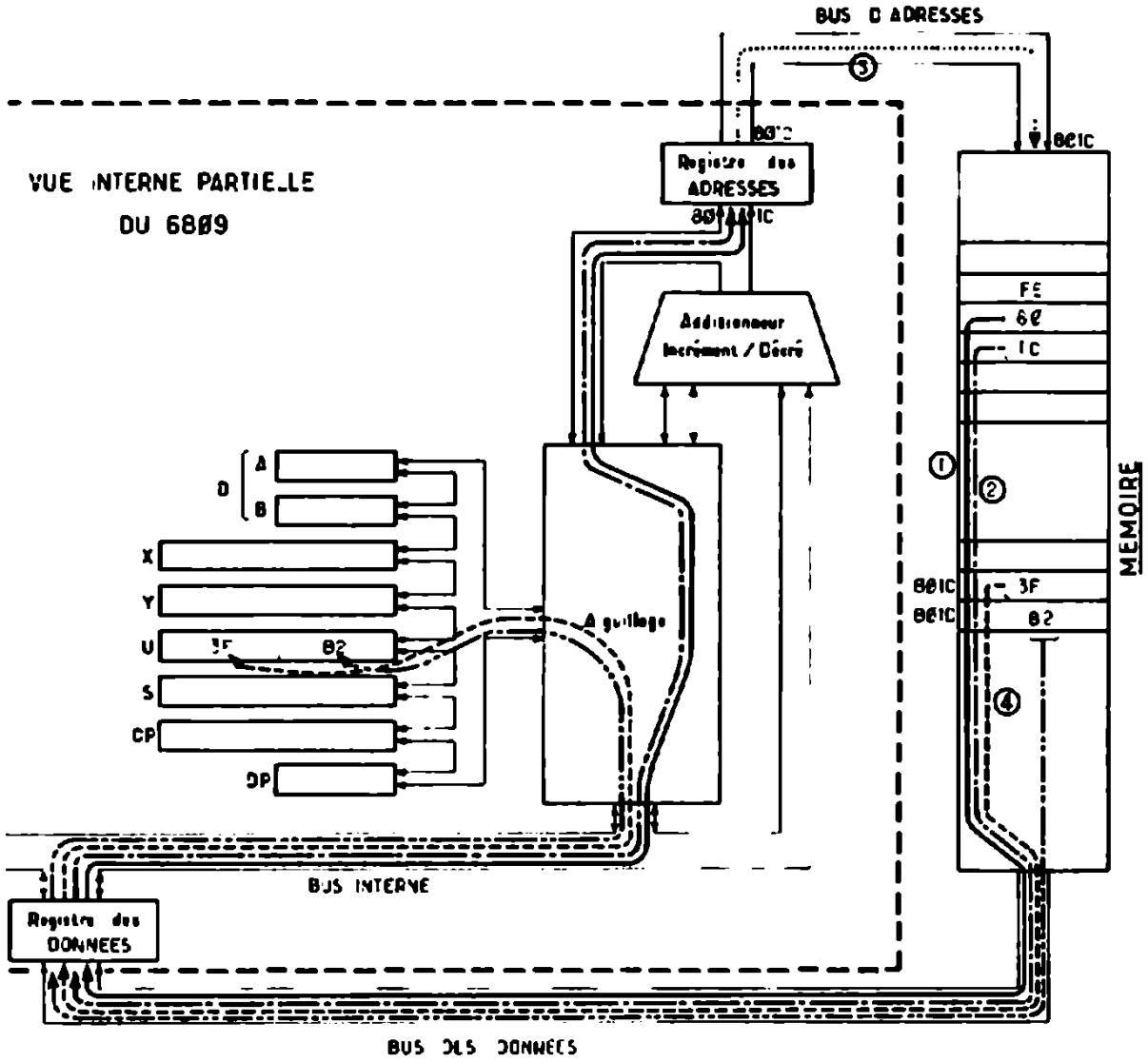


Fig. 18-3. – Adressage étendu.

Reprenons l'exemple précédent ; on écrira :

LDU $ 1C → DE 1C

(Voir fig. 18-4)

Le registre DP contient $8Ø.

L'intérêt de ce mode d'adressage est qu'il est à la fois plus rapide et plus économe en mémoire que l'adressage étendu, à condition toutefois que l'on ait à manipuler plusieurs données se trouvant à la même page car, sinon, le temps passé à charger et recharger DP annule les bénéfices de l'opération.

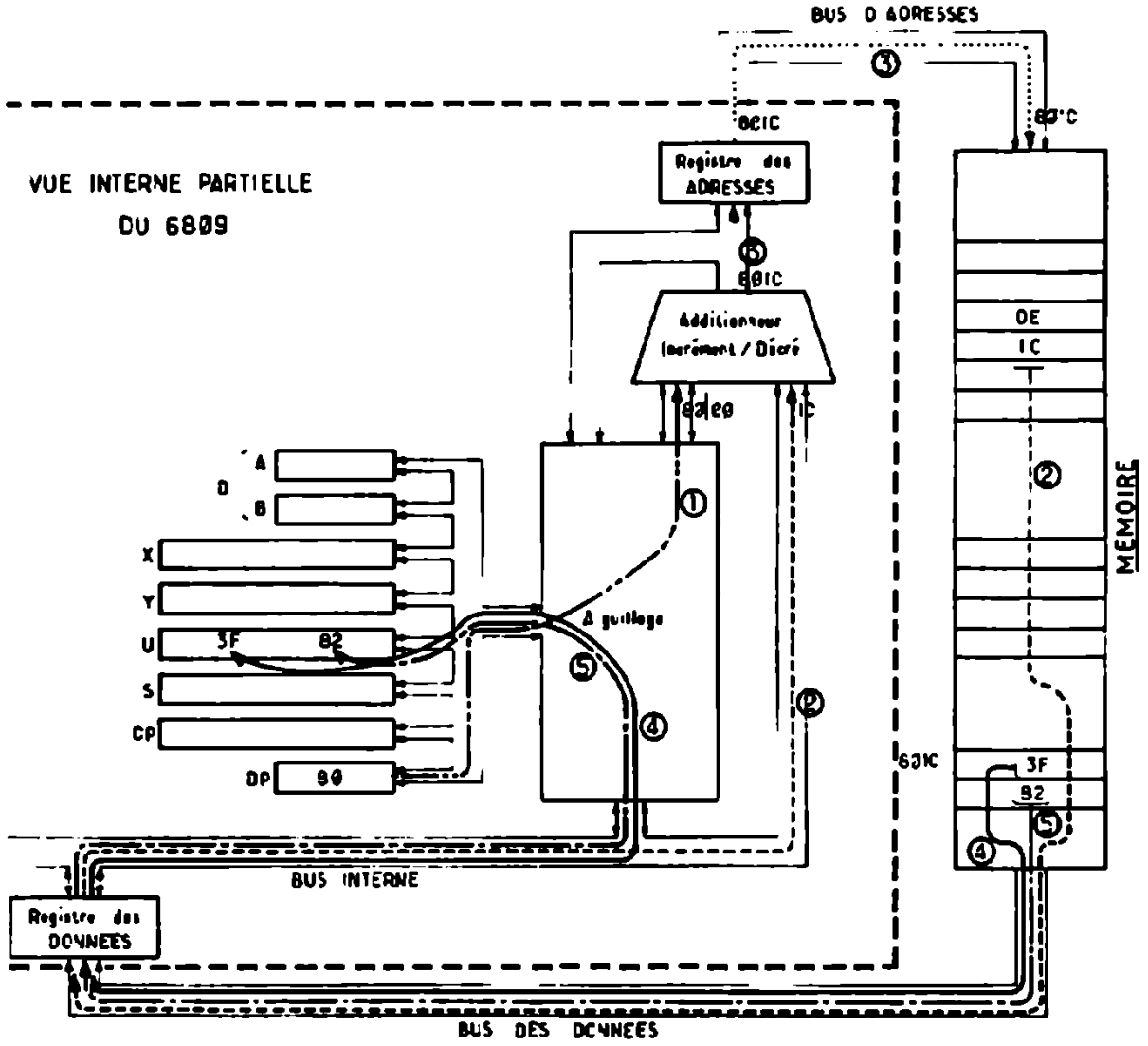


Fig. 18-4. – Adressage direct.

5. Adressage étendu indirect

Dans ce mode d'adressage, la donnée est à une adresse que l'on ne connaît pas directement. Mais on sait où est rangée cette adresse, c'est-à-dire que l'on connaît l'adresse de l'adresse.

Exemple : On veut comparer le contenu de l'accumulateur A avec une valeur rangée à l'adresse Adr. La valeur de cette adresse est elle-même rangée en &H67ØØ, par exemple.

Le code opération est celui du mode indexé suivi de &H9F, soit deux octets dans le cas général (parfois, lorsque l'opération por-

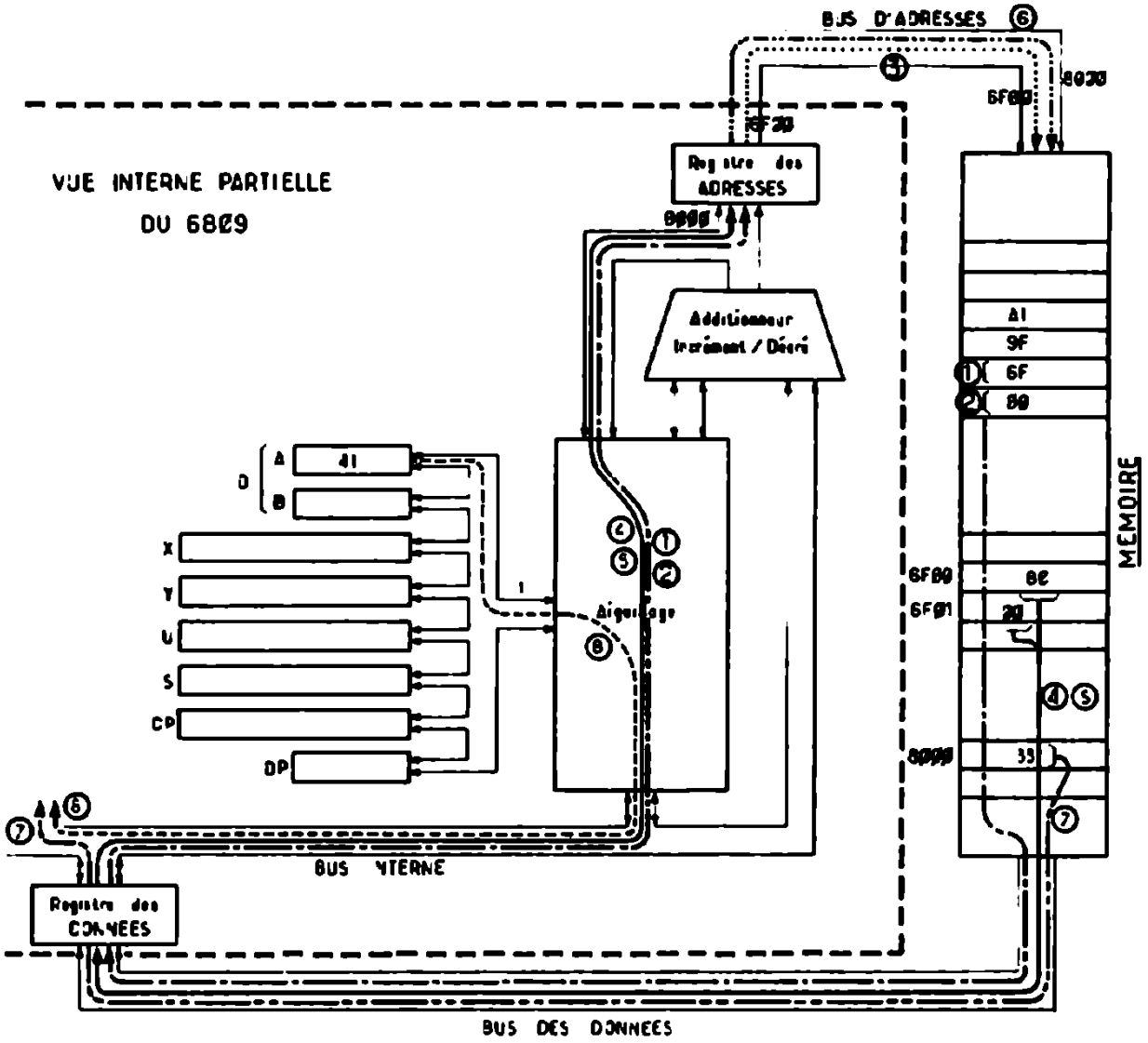


Fig. 18-5. – Adressage étendu indirect.

tera sur des pointeurs, un pré-octet viendra s'ajouter au code opération).

On écrira donc ici :

CMPA [$67ØØ] → A1 9F 6F ØØ

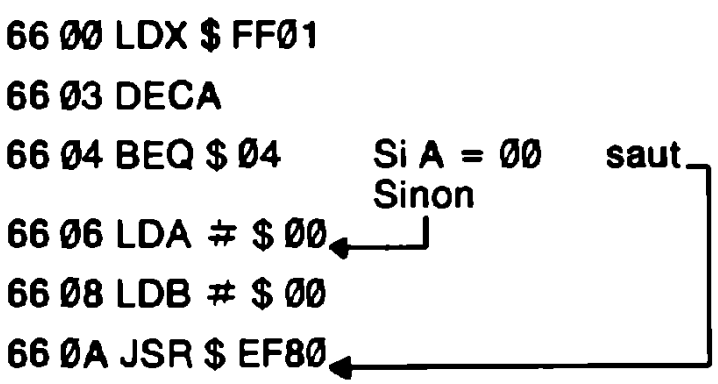
(Voir fig. 18-5)

La procédure sera alors celle de la figure. Les deux crochets indiquent qu'il s'agit d'un adressage indirect.

6. *Adressage relatif court*

Ce mode d'adressage est utilisé uniquement dans les opérations de branchement. Un branchement est réalisé à la suite d'un test, si ce dernier est vrai. Par exemple, si on souhaite sauter quatre lignes de programme si la dernière opération a donné un résultat nul, on dira « Branchement si Egal (à zéro) » soit en anglais « Branch if EQUAL » → le mnémonique BEQ.

Donc on obtient BEQ$Ø4. Cette instruction provoque un saut des quatre lignes qui suivent et le programme ne vient donc lire que la cinquième ligne.



Dans ce mode d'adressage, on ne fournit donc pas l'adresse réelle de destination. Celle-ci est en effet calculée par rapport à la valeur du compteur programme. Si le test conditionnant le branchement est vrai, alors la valeur qui suit le code opération est ajoutée au contenu du compteur programme pour obtenir l'adresse de destination.

Lorsque le microprocesseur a terminé de lire les deux octets de l'instruction de branchement, le compteur programme est déjà pointé sur la case mémoire suivante (par exemple ici &H66Ø6). On en déduit donc la formule de calcul de l'adresse relative :

Adr. Relative = Adr. Destination – (Adr. Brancht + 2)

Le branchement pouvant se faire aussi bien en avant (positif) qu'en arrière (négatif), la valeur relative est donc signée, le bit de signe étant le bit de poids fort.

On ne peut donc faire un saut que de + 127 (&H7F) en avant, et de – 128 (&H8Ø) en arrière. La valeur hexadécimale de branchement en arrière doit donc être donnée en mode complément à 2.

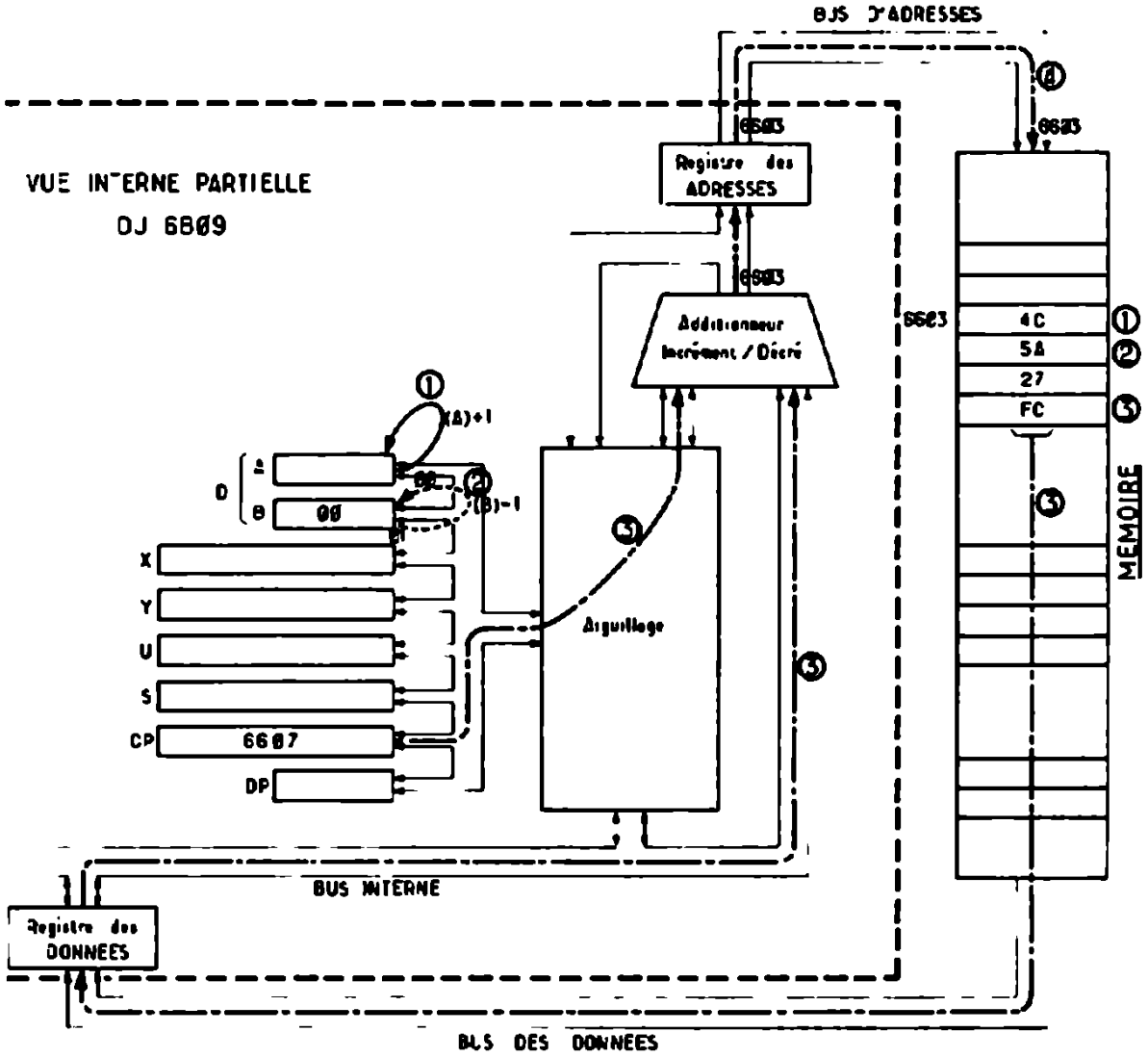


Fig. 18-6. – Adressage relatif court.

Exemple : 66 00 LDX $ F 000

Adr 66 03 INCA 4C

66 04 DECB 5A

66 05 BEQ Adr 27 FC

66 07 JSR $ EF 08

$Adr = COM_2(|\ \&H6603 - \&H6607\ |) = COM_2(\&H04)$

$\&H04 = 00000100$

$COM_2(\&H04) = 11111100 = \&HFC$

Donc on écrira en 6605 BEQ $ FC

7. Adressage relatif long

Ce mode d'adressage, de même principe que le précédent, permet des sauts sur la totalité de la zone mémoire, c'est-à-dire de + 32767 (&H7FFF) en avant, et – 32768 (&H8000) en arrière.

Le code opération contient toujours le pré-octet &H10.

Exemple : BEQ $04 peut aussi s'écrire

LBEQ $0004 → 10270004

8. Adressage indexé

Dans ce mode d'adressage, l'un des registres 16 bits X, Y, U, S, ou le compteur programme, sert d'INDEX pour le calcul de l'adresse effective de la valeur à traiter.

L'adresse de la valeur à traiter est calculée à partir de la valeur contenue dans l'INDEX de référence, choisi par l'utilisateur, et appelé BASE.

On distingue plusieurs possibilités d'indexation.

a) Adressage indexé à déplacement nul

Dans ce mode, la base sélectionnée contient l'adresse où se trouve la valeur à traiter.

Le code opération est suivi d'un post-octet de valeur binaire : 1BB00100 (voir tableau ci-dessous).

Les deux octets B B définissent la base conformément au tableau :

B	B	Registre
0	0	X
0	1	Y
1	0	U
1	1	S

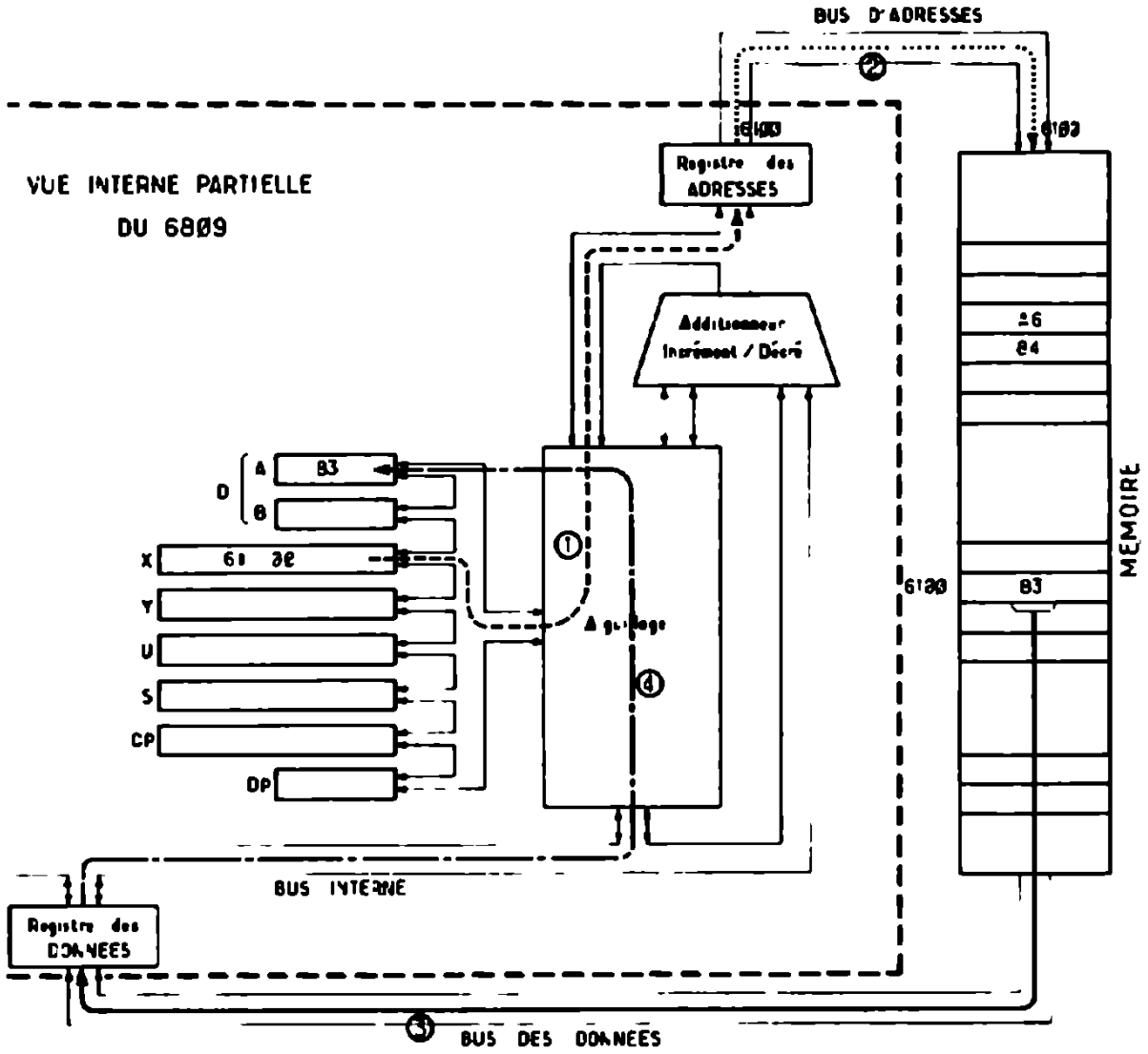


Fig. 18-7. – Adressage indexé à déplacement nul.

Exemple : On veut charger l'accumulateur A avec la valeur qui se trouve en 6100. Le registre X contient l'adresse &H6100.

X = &H6100

LDA 0,X → A6 84

(Voir fig. 18-7)

C'est le mode indexé le plus rapide.

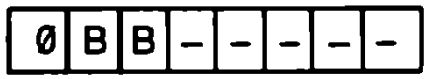
b) Adressage indexé à déplacement constant

Dans ce mode d'adressage, un déplacement est ajouté à la base choisie pour obtenir l'adresse de la donnée. Ce déplacement peut être codé sur 5, 8 et 16 bits.

Dans chaque cas, le bit de poids fort est le bit de signe. Les valeurs négatives sont en mode complément à 2.

– **Déplacement codé sur 5 bits :** dans ce cas, on peut se déplacer par rapport à la base de – 16 à + 15. La valeur du déplacement est contenue dans le post-octet :

Signe



Base Déplacement

Pour comparer B à une valeur située à l'adresse (X)+2, on écrira : CMPB 2,X → E102

Pour réaliser un OU logique entre A et une valeur située à l'adresse (U)–3, on écrira : ORA –3,U → AA 5D

– **Déplacement codé sur 8 bits :** dans ce cas, on peut se déplacer par rapport à la base de – 128 à + 127. La valeur du déplacement suit le post-octet. Ce dernier s'écrit : 1BB01000.

Pour stocker D à l'adresse (S)+64, on écrira :

STD +64,S → ED E8 40

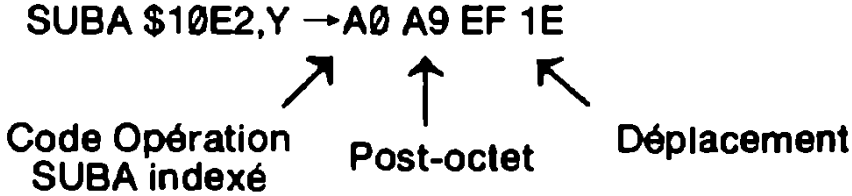
(Voir fig. 18-8)

– **Déplacement codé sur 16 bits :** dans ce cas, on peut se déplacer par rapport à la base de – 32768 à + 32767. La valeur du déplacement est codée sur deux octets qui suivent le post-octet.

Ce dernier s'écrit : 1BB01001.

Pour soustraire à A une valeur située à l'adresse (Y)–&H10E2, on écrira :

SUBA $10E2,Y →A0 A9 EF 1E



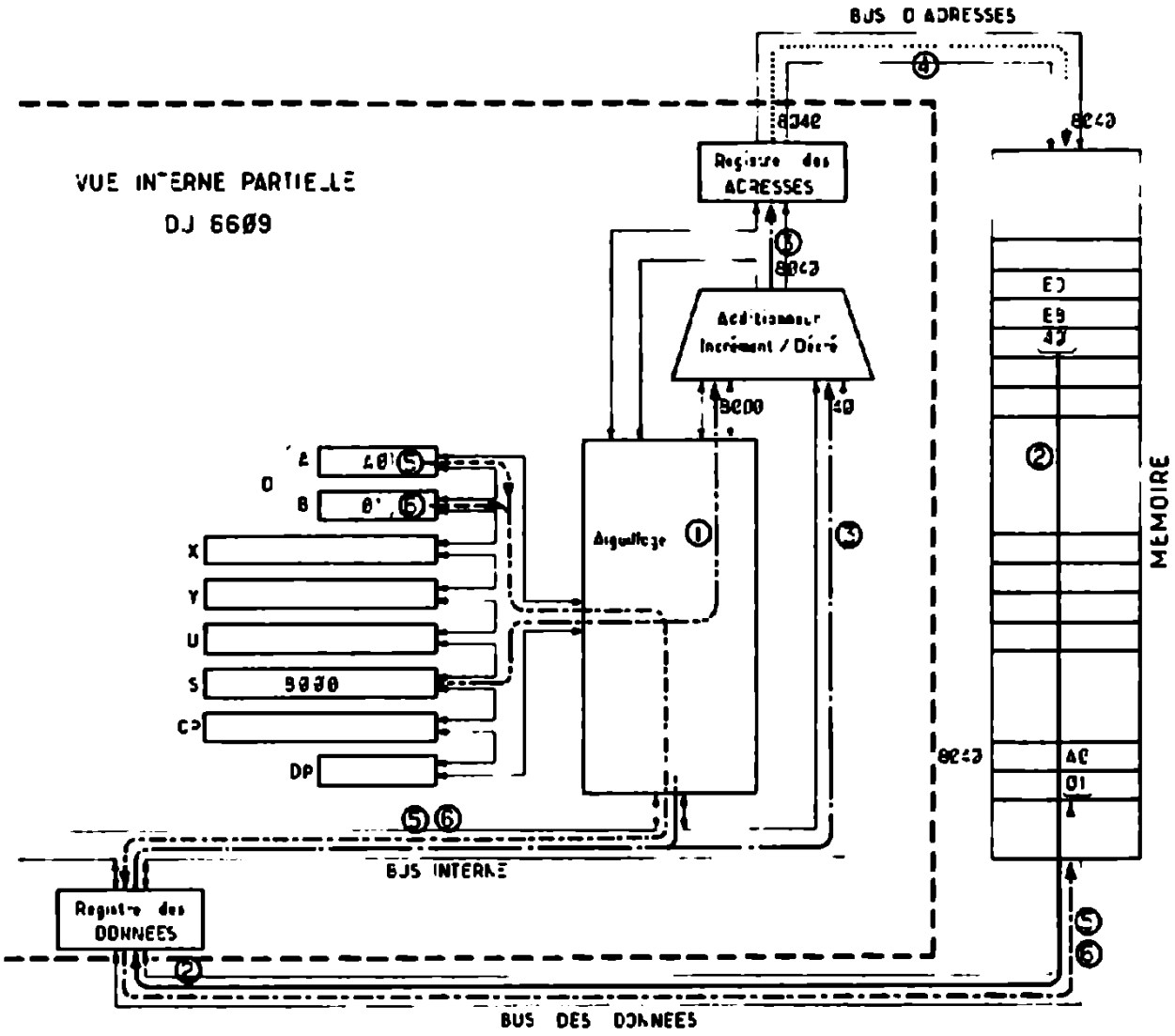


Fig. 18-8. – Adressage indexé – déplacement constant sur 8 bits.

c) Adressage indexé avec auto-incrémentation/décrémentation

Dans ce mode d'adressage, la base contient l'adresse de destination.

En auto-incrémentation, la valeur contenue dans la base est incrémentée de un ou de deux *après* avoir effectué l'opération.

En auto-décrémentation, la valeur contenue dans la base est décrémentée de un ou de deux *avant* d'avoir effectué l'opération.

Le post-octet a l'une des formes ci-dessous.

1 B B 0 0 0 0 0	,BASE +
1 B B 0 0 0 0 1	,BASE++
1 B B 0 0 0 1 0	,– BASE
1 B B 0 0 0 1 1	,– – BASE

Ce mode d'adressage est particulièrement utile pour la gestion de tableaux de valeurs.

– Lorsque les valeurs de données ont 8 bits, on travaille en mode auto-incrémentation/décrémentation de un.

– Lorsque les valeurs des données ont 16 bits, on incrémente ou décrémente de deux.

On peut utiliser les registres X et Y comme des piles logicielles ainsi que U et S.

d) Adressage indexé – Déplacement accumulateur

Dans ce mode d'adressage, l'adresse effective est obtenue en ajoutant à la base le contenu de l'un quelconque des accumulateurs A, B ou D.

Le post-octet est alors le suivant :

1 B B 0 1 0 1 1 pour Acc D
1 B B 0 0 1 1 0 pour Acc A
1 B B 0 0 1 0 1 pour Acc B

Exemple : On veut charger l'accumulateur A avec la valeur qui se trouve à l'adresse obtenue en ajoutant le contenu de l'accumulateur B à celui du registre Y. On écrit donc :

LDA B,Y → A6 A5

On remarquera que, dans ce cas, l'accumulateur B (8 bits) vient s'ajouter à l'octet de poids faible du registre Y (16 bits).

(Voir fig. 18-9)

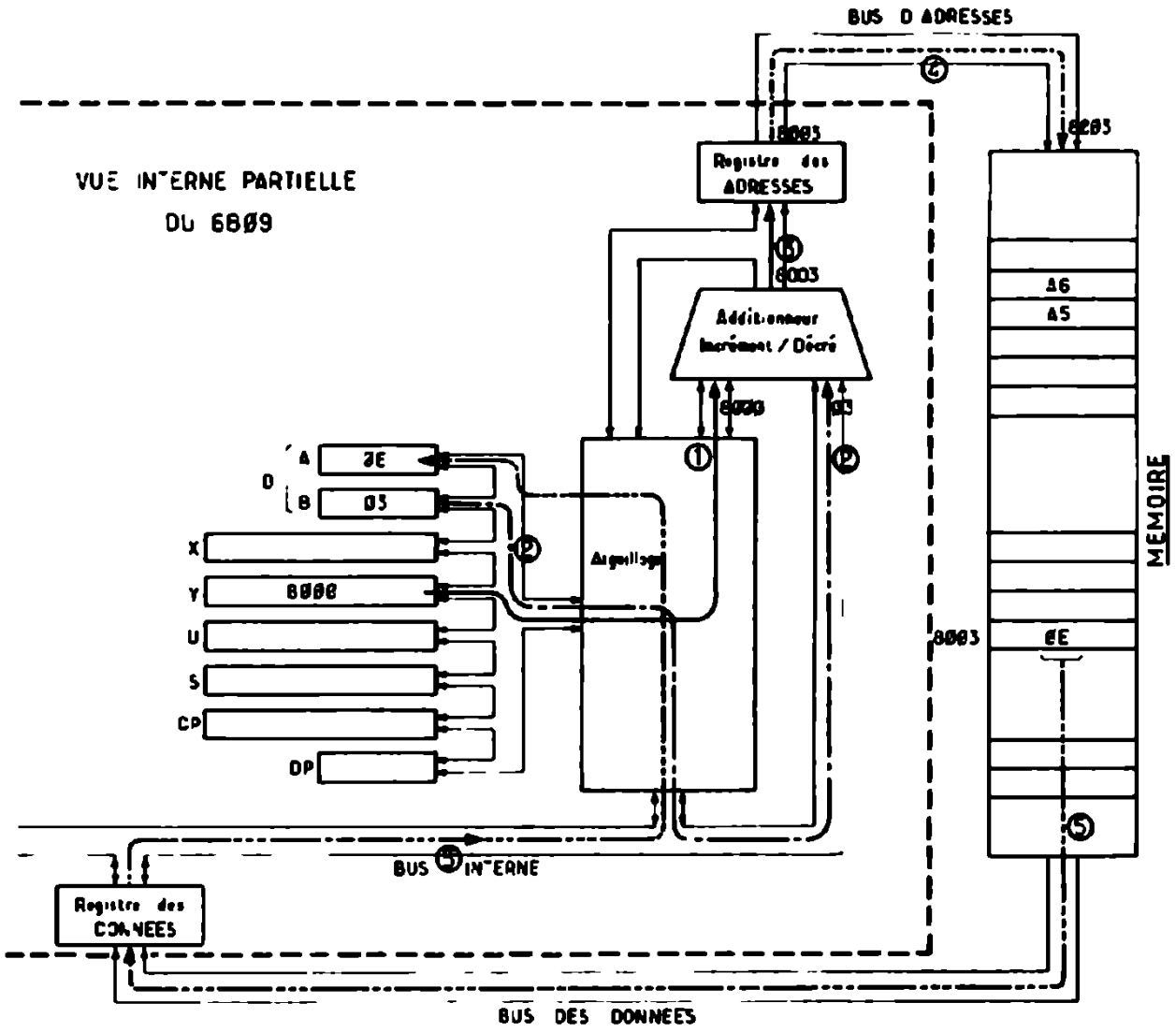


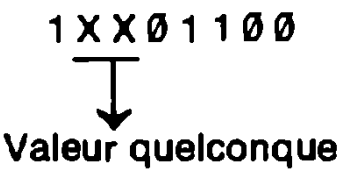
Fig. 18-9. – Adressage Indexé – déplacement accumulateur.

e) Adressage indexé – Base = compteur-programme

Dans ce mode d'adressage, le compteur programme est la base de calcul de l'adresse. Il s'agit donc en fait d'un adressage relatif qui peut avoir deux modes, long ou court.

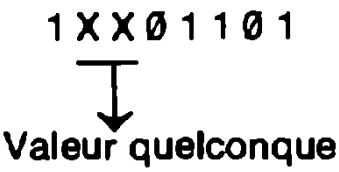
– Déplacements sur 8 bits :

Dans ce cas, on peut se déplacer de – 128 à + 127 par rapport au compteur programme. L'octet de déplacement suit le post-octet qui s'écrit :



– **Déplacement sur 16 bits :**

Dans ce cas, on peut se déplacer de – 32768 à + 32767 par rapport au compteur-programme. Les deux octets du déplacement suivent le post-octet qui s'écrit :



9. Adressage indexé-indirect

Dans ce type d'adressage, la valeur calculée à partir de la base et du déplacement, contient cette fois l'adresse de l'adresse de la valeur à traiter.

Une indirection s'ajoute donc à l'indexation.

On obtient ce mode d'adressage en faisant passer à 1 le bit 4 du post-octet.

L'indirection n'est pas possible avec les modes indexés suivants :

– déplacement constant sur 5 bits
– auto-incrémentation/décrémentation de un.

Exemple : On veut charger A avec une valeur dont l'adresse se trouve à l'adresse obtenue en ajoutant &H1Ø au registre X. On écrit :

LDA [1Ø,X] → A6 98 1Ø.

(Voir fig. 18-10)

10. L'instruction LEA (Load Effective Address)

Cette instruction, qui ne s'utilise qu'en mode indexé, ne concerne que les registres d'index X, Y, U et S. Elle permet de charger, dans le registre choisi, une valeur calculée à partir de l'indexation, l'index pouvant être un autre registre.

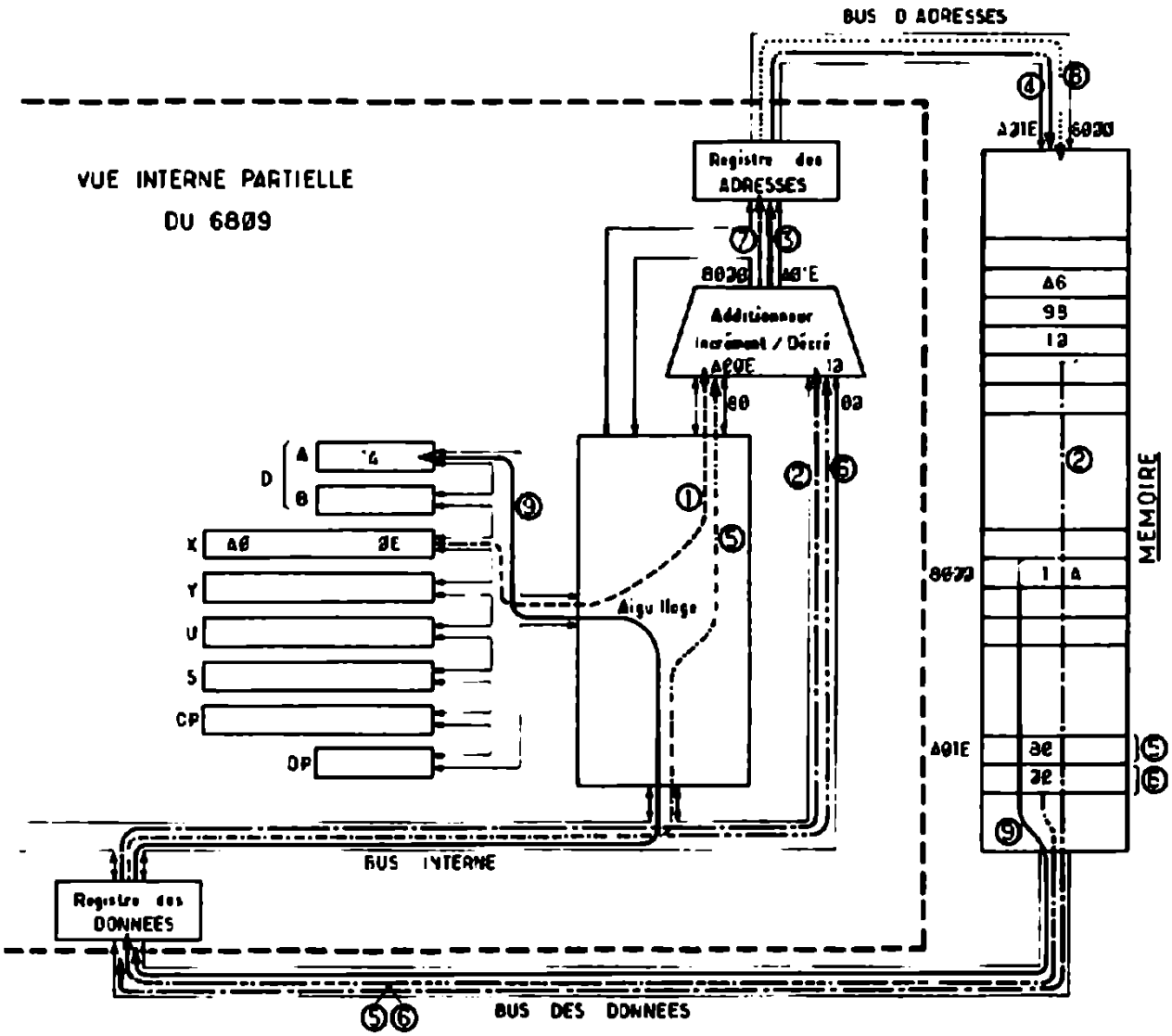


Fig. 18-10. Adressage Indexé Indirect.

Exemple :

On peut écrire LEAY $ Ø8,X → 31 Ø8

Ce qui revient à mettre dans Y la valeur qui se trouve à l'adresse calculée en ajoutant Ø8 au contenu de X.

L'indirection est applicable à cette instruction.

Exemple :

Si l'on écrit LEAU [1F82,U] → 33 D9 1F 82.

On charge U avec la valeur dont l'adresse se trouve à l'adresse calculée en ajoutant 1F82 au contenu de U.

Chapitre 19

LE JEU D'INSTRUCTIONS DU 6809

Tableau 19-1. – 8-Bit Accumulator and Memory Instructions.

Instruction	Description
ADCA, ADCB	Add memory to accumulator with carry
ADDA, ADDB	Add memory to accumulator
ANDA, ANDB	And memory with accumulator
ASL, ASLA, ASLB	Arithmetic shift of accumulator or memory left
ASR, ASRA, ASRB	Arithmetic shift of accumulator or memory right
BITA, BITB	Bit test memory with accumulator
CLR, CLRA, CLRB	Clear accumulator or memory location
CMPA, CMPB	Compare memory from accumulator
COM, COMA, COMB	Complement accumulator or memory location
DAA	Decimal adjust A accumulator
DEC, DECA, DECB	Decrement accumulator or memory location
EORA, EORB	Exclusive or memory with accumulator
EXG R1, R2	Exchange R1 with R2 (R1, R2 = A, B, CC, DP)
INC, INCA, INCB	Increment accumulator or memory location
LDA, LDB	Load accumulator from memory
LSL, LSLA, LSLB	Logical shift left accumulator or memory location
LSR, LSRA, LSRB	Logical shift right accumulator or memory location

Instruction	Description
MUL	Unsigned multiply (A x B →D)
NEG, NEGA, NEGB	Negate accumulator or memory
ORA, ORB	Or memory with accumulator
ROL, ROLA, ROLB	Rotate accumulator or memory left
ROR, RORA, RORB	Rotate accumulator or memory right
SBCA, SBCB	Subtract memory from accumulator with borrow
STA, STB	Store accumulator to memory
SUBA, SUBB	Subtract memory from accumulator
TST, TSTA, TSTB	Test accumulator or memory location
TFR R1, R2	Transfer R1 to R2 (R1, R2 =A, B, CC, DP)

NOTE : A, B, CC or DP may be pushed to (pulled from) either stack with PSHS, PSHU (PULS, PULU) instructions.

Tableau 19-2. – 16-Bit Accumulator and Memory Instructions.

Instruction	Description
ADDD	Add memory to D accumulator
CMPD	Compare memory from D accumulator
EXG D, R	Exchange D with X, Y, S, U or PC
LDD	Load D accumulator from memory
SEX	Sign Extend B accumulator into A accumulator
STD	Store D accumulator to memory
SUBD	Subtract memory from D accumulator
TFR D, R	Transfer D to X, Y, S, U or PC
TFR R, D	Transfer X, Y, S, U or PC to D

NOTE : D may pushed (pulled) to either stack with PSHS, PSHU (PULS, PULU) instructions.

Tableau 19-3. – Index/Stack Pointer Instructions.

Instruction	Description
CMPS, CMPU	Compare memory from stack pointer
CMPX, CMPY	Compare memory from index register
EXG R1, R2	Exchange D, X, Y, S, U or PC with D, X, Y, S, U or PC
LEAS, LEAU	Load effective address into stack pointer
LEAX, LEAY	Load effective address into index register
LDS, LDU	Load stack pointer from memory
LDX, LDY	Load index register from memory
PSHS	Push A, B, CC, DP, D, X, Y, U or PC onto hardware stack
PSHU	Push A, B, CC, DP, D, X, Y, X or PC onto user stack
PULS	Pull A, B, CC, DP, D, X, Y, U or PC from hardware stack
PULU	Pull A, B, CC, DP, D, X, Y, S or PC from hardware stack
STS, STU	Store stack pointer to memory
STX, STY	Store index register to memory
TFR R1, R2	Transfer D, X, Y, S, U or PC to D, X, Y, S, U or PC
ABX	Add B accumulator to X (unsigned)

Tableau 19-4. – Branch Instructions.

Instruction	Description
	SIMPLE BRANCHES
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BMI, LBMI	Branch if minus
BPL, LBPL	Branch if plus
BCS, LBCS	Branch if carry set

Instruction	Description
BCC, LBCC	Branch if carry clear
BVS, LBVS	Branch if overflow set
BVC, LBVC	Branch if overflow clear
	SIGNED BRANCHES
BGT, LBGT	Branch if greater (signed)
BVS, LBVS	Branch if invalid twos complement result
BGE, LBGE	Branch if greater than or equal (signed)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLE, LBLE	Branch if less than or equal (signed)
BVC, LBVC	Branch if valid twos complement result
BLT, LBLT	Branch if less than (signed)
	UNSIGNED BRANCHES
BHI, LBHI	Branch if higher (unsigned)
BCC, LBCC	Branch if higher or same (unsigned)
BHS, LBHS	Branch if higher or same (unsigned)
BEQ, LBEQ	Branch if equal
BNE, LBNE	Branch if not equal
BLS, LBLS	Branch if lower or same (unsigned)
BCS, LBCS	Branch if lower (unsigned)
BLO, LBLO	Branch if lower (unsigned)
	OTHER BRANCHES
BSR, LBSR	Branch to subroutine
BRA, LBRA	Branch always
BRN, LBRN	Branch never

Tableau 19-5. – Miscellaneous instructions.

Instruction	Description
ANDCC	AND condition code register
CWAI	AND condition code register, then wait for interrupt
NOP	No operation
ORCC	OR condition code register
JMP	Jump
JSR	Jump to subroutine
RTI	Return from interrupt
RTS	Return from subroutine
SWI, SWI2, SWI3	Software interrupt (absolute indirect)
SYNC	Synchronize with interrupt line

5 Le moniteur et la programmation en langage machine

Chapitre 20

PROGRAMMATION EN LANGAGE MACHINE

1. Mini éditeur « LANGAGE »

Pour pouvoir réaliser avec facilité ses programmes en langage machine, il faut disposer d'un logiciel minimum qui permette :

1° d'implanter de l'hexadécimal à une adresse donnée ;

2° de changer un octet à une adresse (ou entre deux adresses) donnée(s) et de le remplacer par un autre ;

3° de déplacer un bloc mémoire pour le réimplanter à partir d'une adresse donnée ;

4° de lister sur l'écran un bloc mémoire.

Le programme ci-joint vous permet d'obtenir ces quatre fonctions. A la mise en route (RUN), votre écran devient noir et vous voyez apparaître une flèche → bleu ciel qui vous indique que vous pouvez taper une commande.

Vous disposez de cinq commandes :

1re commande : A HHHH [ENTREE]

Exemple : A 6000

Cette instruction vous permet de rentrer des codes hexadécimaux à partir de l'adresse hexadécimale 6000.

Attention : Il faut toujours laisser un espace après la lettre de commande A avant de taper l'adresse.

La commande apparaît en bleu ciel.

Dès que vous avez tapé sur [ENTREE], le programme vous renvoie l'adresse **HHHH** (en rouge) et attend vos codes hexadécimaux.

Tapez alors vos codes à la suite les uns des autres, sans espace. C'est le programme qui séparera les octets.

A la fin d'une ligne, tapez [ENTREE]. Le logiciel reviendra à la ligne et vous indiquera (en rouge) le nouveau numéro de ligne. Vous pourrez alors :

– soit retaper des codes hexa comme précédemment,

– soit retaper sur [ENTREE] pour repasser sous contrôle des commandes. Dans ce cas, la flèche bleue réapparaît.

2° commande : C HHHH/OO/NN

Cette commande permet de **changer** l'octet hexadécimal **OO** à la ligne **HHHH** et de le remplacer par la valeur hexadécimale **NN.**

Si OO n'existe pas à la ligne HHHH, un message d'erreur « Erreur de contenu mémoire » est envoyé sur l'écran, et la flèche réapparaît. Sinon la correction est faite et la nouvelle ligne HHHH apparaît avec son nouveau contenu NN.

Cette commande permet également de remplacer l'octet OO par NN partout où cet octet se trouve entre les deux lignes HHHH et JJJJ. Il faut alors commander :

C HHHH-JJJJ/OO/NN

3° commande : D HHHH-JJJJ/KKKK

Cette commande déplace le bloc mémoire allant de **HHHH** à **JJJJ** (hexa) pour le réimplanter à partir de l'adresse **KKKK.**

4° commande : L HHHH-JJJJ

Cette commande permet de lister sur l'écran toutes les valeurs comprises de l'adresse HHHH à JJJJ.

5° commande : F

C'est une commande de **fin** qui vous renvoie au Basic.

Vous pouvez alors sauver un programme machine, implanté de 6000 à 6100, par exemple, en faisant :

SAVEM « NOM », &H6000, &H6100, &H6000

2. Listing de « LANGAGE »

```
10 '                  ****************
20 '                  *              *
30 '                  *   LANGAGE    *
40 '                  *              *
50 '                  ****************
60 '
70 '
80 CLS:SCREEN2,0,0:CLEAR,&H5FFF,1
90 DEFGR$(0) = 0,8,4,2,255,2,4,8
100 COLOR6:PRINT:PRINTGR$(0)::LINEINPUTA$:L$=LEFT$(A$,1)
:A$=L$+L$+A$:PRINT:COLOR2
110 IFL$="C"GOTO290
120 IFL$="F"THENEND
130 IFL$="L"GOTO430
140 IFL$="D"GOTO540
150 IFL$="A"GOTO650
160 COLOR1:PRINT"Instruction inexistante.":COLOR2:GOTO10
0
170 FORN=1TO4
180 D$=MID$(A$,I+N,1):GOSUB220
190 NEXTN
200 NL=D(1)*4096+D(2)*256+D(3)*16+D(4)
210 RETURN
220 IF ASC(D$)<60 THEN D(N)=VAL(D$) ELSE D(N)=ASC(D$)-55
230 RETURN
240 FORN=1TO2
250 D$=MID$(X$,N,1):GOSUB220
260 NEXTN
270 X=D(1)*16+D(2):RETURN
280 E$=INPUT$(1) : RETURN
290 IFMID$(A$,4,1)=" "GOTO300ELSEGOTO320
300 IFMID$(A$,9,1)="/"THENIFMID$(A$,12,1)="/"GOTO330ELSE
GOTO320
310 IFMID$(A$,9,1)="-"THENIFMID$(A$,14,1)="/"THENIFMID$(
A$,17,1)="/"GOTO370ELSEGOTO320
320 COLOR1:PRINT"Erreur de syntaxe.":COLOR2:GOTO100
```

```
330 PRINT:I=4:GOSUB170
340 X$=MID$(A$,10,2):GOSUB240:IFX=PEEK(NL)GOTO350ELSEGOT
0360
350 X$=MID$(A$,13,2):GOSUB240:POKENL,X:COLOR3:GOSUB740:P
RINTHX$+" ";:COLOR2:GOSUB810:PRINTHX$:GOTO100
360 COLOR1:PRINT"Erreur de contenu memoire.":COLOR2:GOTO
100
370 I=4:GOSUB170:ND=NL
380 I=9:GOSUB170:NF=NL:X$=MID$(A$,15,2):GOSUB240:Y=X:X$=
MID$(A$,18,2):GOSUB240
390 FORNL=ND TO NF
400 IFY=PEEK(NL)THENPOKENL,X:COLOR3:GOSUB740:PRINTHX$+"
";:COLOR2:GOSUB810:PRINTHX$
410 NEXTNL
420 GOTO100
430 IFMID$(A$,4,1)=" "THENIFMID$(A$,9,1)="-"GOTO440ELSEC
OLOR1:PRINT"Erreur de syntaxe":COLOR2:GOTO100
440 I=4:GOSUB170:ND=NL:I=9:GOSUB170:NF=NL
450 FOR NL=ND TO NF STEP8
460 COLOR3:GOSUB740:PRINTHX$+" ";:COLOR2
470 FORI=0TO7
480 X=PEEK(NL+I):GOSUB810:X$=HX$:IFLEN(X$)=1 THENX$="0"+
X$
490 PRINTX$+" ";
500 NEXTI
510 PRINTCHR$(13)
520 NEXTNL
530 GOTO100
540 IFMID$(A$,4,1)=" "THENIFMID$(A$,9,1)="-"THENIFMID$(A
$,14,1)="/"GOTO550ELSECOLOR1:PRINT"Erreur de syntaxe.":C
OLOR2:GOTO100
550 PRINT:I=4:GOSUB170:ND=NL
560 I=9:GOSUB170:NF=NL
570 I=14:GOSUB170:NA=NL:NB=NL:NC=NB+NF-ND
580 IFNA>ND THENNA=NA+NF-ND:GOTO620
590 FORNL=ND TO NF
600 POKENA,PEEK(NL):NA=NA+1
610 NEXTNL:ND=NB:NF=NC:GOTO450
620 FORNL=NF TO ND STEP-1
```

```
630 POKENA,PEEK(NL):NA=NA-1
640 NEXTNL:ND=NB:NF=NC:OPEN"O",#1,"SCRN:":GOTO450
650 IFMID$(A$,4,1)=" "THENGOTO660ELSECOLOR1:PRINT"Erreur
 de syntaxe.":COLOR2:GOTO100
660 I=4:GOSUB170
670 COLOR1:GOSUB740:PRINTHX$+" ";:COLOR2
680 GOSUB280
690 IFE$=CHR$(13)GOTO730ELSEX$=E$:PRINTE$;
700 GOSUB280:X$=X$+E$:PRINTE$+" ";:GOSUB240:POKENL,X:NL=
NL+1:X$=""
710 GOSUB280
720 IFE$=CHR$(13)THENPRINTE$:GOTO670ELSEX$=E$:PRINTE$;:G
OTO700
730 PRINT:GOTO100
740 HX$="":AD=NL
750 FORJ=3TO0STEP-1
760 S(J)=INT(AD/(16^J)):AD=AD-S(J)*(16^J)
770 IF S(J)<10 THEN S$(J)=CHR$(48+S(J))ELSE S$(J)=CHR$(5
5+S(J))
780 HX$=HX$+S$(J)
790 NEXTJ
800 RETURN
810 HX$="":V=X
820 FORJ=1TO0STEP-1
830 D(J)=INT(V/(16^J)):V=V-D(J)*(16^J)
840 IF D(J)<10 THEN D$(J)=CHR$(48+D(J))ELSE D$(J)=CHR$(5
5+D(J))
850 HX$=HX$+D$(J)
860 NEXTJ
870 RETURN
```

Chapitre 21

LE MONITEUR DU MO5

Le moniteur du MO5 occupe les 4 K-octets compris entre $F000 et $FFFF. Ce moniteur contient un certain nombre de sous-programmes que l'utilisateur peut appeler à partir d'un programme écrit en langage machine.

Les principales fonctions de ces sous-programmes sont :

- la gestion du clavier
- la gestion de l'écran
- la gestion de l'interface communication
- la gestion de l'extension jeux
- la gestion du LEP
- la gestion du contrôleur de disquette
- la gestion du crayon optique.

1. Accès aux sous-programmes du moniteur

Pour accéder à un sous-programme du moniteur, il faut utiliser l'instruction d'interruption logicielle « SWI » (code hexa &H3F), suivi d'un code (octet compris entre &H00 et &H3A) qui sélectionne le sous-programme.

Ces instructions sont équivalentes à un JSR vers le sous-programme choisi.

Puisqu'une instruction SWI est utilisée, il y a automatiquement sauvegarde sur la pile système de l'état du 6809. En outre, le registre de page direct DP est chargé avec la valeur &H20, ce qui permettra de travailler avec les registres de la page zéro du moniteur (de &H2000 à &H20FF) en mode direct, et le pointeur de pile U est chargé avec l'adresse du PIA système, c'est-à-dire &HA7C0.

En revenant du sous-programme, tous les registres sont dépilés, sauf le CCR qui conserve la valeur courante et évidemment ceux des accumulateurs (B) ou registres (X,Y) qui reviennent avec un résultat.

Exemple : pour accéder à la routine de gestion de l'écran PUTC, il faudra faire :

```
5000 3F                  SWI
6001       02            FCB     2
```

2. Utilisation des registres de la page Ø

Les registres de la page Ø occupent un, deux ou trois octets. Lorsque ces registres servent au passage de paramètres, ils occupent un ou deux octets suivant que ces paramètres sont de 8 ou 16 bits.

Les pointeurs occupent, eux, trois octets :

– les deux premiers octets contiennent l'adresse pointée (16 bits)

– le troisième octet est un flag :
  – mis à Ø si le pointeur contient l'adresse standard
  – mis à 1 si le pointeur contient l'adresse définie par l'utilisateur.

Lorsque l'on charge un pointeur avec sa propre adresse, il est **obligatoire** de mettre le drapeau (3e byte) à 1.

Si l'on restaure l'adresse standard, il **faut** le remettre à Ø.

Les pointeurs de la page Ø (voir Annexe 4) sont :

SWI1PT, TIMEPT, IRQPT, FIRQPT, SIMUL, CHRPTR, USERAF, GENPTR.

3. Gestion de tables

La mémoire du MO5 est divisée en segments de 2 K-octets à partir de &HØØØØ. Les divers segments sont donc :

Segment n° 1 &HØØØØ à &HØ7FF

Segment n° 2 &HØ8ØØ à &HØFFF

Segment n° 3 &H1ØØØ à &H17FF

Segment n° 4 &H18ØØ à &H1FFF

etc.

Toutes les tables créées par l'utilisateur ne doivent pas excéder 2 K-octets et doivent être contenues dans un seul et même segment.

4. *Sous-programme « MENU »*

Ce sous-programme renvoie au point d'entrée du Basic (ou de la ROM d'application).

```
6000                    ORG     $6000

6000 3F                 SWI
6001       00           FCB     0

           0000         END

00000 Total Errors
```

On y accède par :

Exemple :

A l'aide de « LANGAGE », rentrer en &H6000 le programme suivant :

→ A 6000
6000 3F 00
6002
→ F

Puis faites :

EXEC &H6000

5. *Les sous-programmes graphiques*

1° Le sous-programme « PLOT »

On y accède par :

```
6000                    ORG     $6000

6000 3F                 SWI
6001       10           FCB     $10

           0000         END

00000 Total Errors
```

Les variables d'entrées sont :

- les registres X et Y du 6809
- les registres FORME &H2029
  CHDRAW &H2036
  COLOUR &H202B

de la page zéro du moniteur.

• *En mode graphique*, ce sous-programme allume le point de l'écran dont les coordonnées sont contenues dans les registres X (0 à 319) et Y (0 à 199).

La couleur de ce point est celle codée dans le registre FORME :

- de – 16 à – 1 pour une couleur FOND
- de 0 à 15 pour une couleur FORME.

Si le bit 4 du registre STATUS (&H2019) est à 1, la couleur ne sera pas mise.

Le registre CHDRAW doit être à zéro (&H00).

Les codes des couleurs sont les suivants :

COULEUR	FORME	FOND
NOIR	00	FF
ROUGE	01	FE
VERT	02	FD
JAUNE	03	FC
BLEU	04	FB
MAGENTA	05	FA
CYAN	06	F9
BLANC	07	F8
GRIS	08	F7
ROSE	09	F6
VERT CLAIR	0A	F5
JAUNE CLAIR	0B	F4
BLEU CIEL	0C	F3
ROSE PARME	0D	F2
CYAN CLAIR	0E	F1
ORANGE	0F	F0

• *En mode caractère*, ce sous-programme écrit le caractère dont le code ASCII est rangé dans le registre CHDRAW, dans la case

de l'écran dont les coordonnées sont contenues dans les registres X (1 à 40) et Y (Ø à 24).

La couleur du caractère est fixée par le contenu du registre COLOUR.

Ce registre contient les 8 bits de code suivants :

FORME	FOND
P B V R	P B V R

P = 1 → teinte « pastel »

B = 1 → bleu

V = 1 → vert

R = 1 → rouge

Exemple : un caractère BLEU sur fond ROSE sera obtenu si COLOUR contient Ø1ØØ1ØØ1, c'est-à-dire &H49.

On peut accéder au mode caractère de PLOT en tapant le code &H12 après le SWI. On dit qu'on appelle alors la routine CHPL.

Application : on veut écrire le mot « BONJOUR » en BLEU sur fond ROSE à partir de la case 3,8.

```
            202B COLOUR EQU    $202B
            2036 CHDRAW EQU    $2036
            0010 PLOT   EQU    $10

5000                    ORG    $6000

6000 86     49   DEBUT  LDA    #$49
6002 B7     202B       STA    COLOUR
6005 CE     601D       LDU    #TABLE
6008 8E     0003       LDX    #3
600B 108E   0008       LDY    #8

600F E6     C0   BOUCLE LDB    ,U+
6011 C1     04         CMPB   #4
6013 27     07         BEQ    FIN
6015 F7     2036       STB    CHDRAW
6018 3F                SWI            }
6019        10         FCB    PLOT    }
```

```
601A 20    F3          BRA     BOUCLE

601C 39          FIN   RTS

601D       42    TABLE FCC     /BONJOUR/
           4F
           4E
           4A
           4F
           55
           52
6024       04          FCB     4

           0000        END

00000 Total Errors
```

Commentaires

L'adresse de début de programme est fixée en $6000.

N° ligne	Commentaire
6000	On charge l'accumulateur A du 6809 avec la valeur $49.
6002	On range ce contenu dans le registre COLOUR, ce qui a pour effet de définir des caractères BLEUS sur fond ROSE.
6005	On charge le pointeur de pile U avec l'adresse de début d'une TABLE qui contient les codes ASCII de /BONJOUR/ et se termine par le code 4 (EOT = End of Text).
6008	Le registre d'index X est chargé avec le numéro de colonne de départ du texte à écrire (soit ici 3).
6008	Le registre d'index Y est chargé avec le numéro de ligne.
600F	On charge B avec la valeur qui se trouve à l'adresse pointée par U, puis on auto-incrémente U (U devient U + 1).
6011	On regarde si B contient la valeur 4 de fin de texte.

6Ø13	Si oui, alors on a fini, et on se branche vers FIN ; sinon, on continue en séquence.
6Ø15	On range le code ASCII contenu dans B dans le registre CHDRAW.
6Ø18	On appelle la routine PLOT, grâce à une interruption SWI.
6Ø1A	On retourne en 6ØØF lire le code suivant.
6Ø1C	C'est la fin, on sort par un Return from Subroutine.
6Ø1D	On range les codes ASCII de /BONJOUR/, puis 4.

Il faudra donc taper avec « LANGAGE » :

6ØØØ	86	49
6ØØ2	B7	202B
6ØØ5	CE	6Ø1D
6ØØ8	8E	ØØØ3
6ØØB	1Ø8E	ØØØ8
6ØØF	E6	CØ
6Ø11	C1	Ø4
6Ø13	27	Ø7
6Ø15	F7	2Ø36
6Ø18	3F	
6Ø19	1Ø	
6Ø1A	2Ø	F3
6Ø1C	39	
6Ø1D	42	4F 4E 4A 4F 55 52 Ø4
6Ø25	ENTREE	

→ F

Puis faire EXEC &H6ØØØ

2° Le sous-programme « DRAW »

On y accède en tapant &HØE après le code &H3F de SWI. Les variables d'entrées sont :

- les registres X et Y du 68Ø9
- les registres PLOTX
  PLOTY
  CHDRAW &H2Ø36

FORME &HØ29
COLOUR &H2Ø2B

de la page zéro du moniteur.

● *En mode graphique*, ce sous-programme tire un trait graphique entre la position courante du curseur point (contenue dans PLOTX et PLOTY) et le point dont les coordonnées sont contenues dans X (Ø à 319) et Y (Ø à 199).

Les contenus des registres FORME, STATUS et CHDRAW sont les mêmes que pour PLOT.

● *En mode caractère*, ce sous-programme tire un trait du caractère contenu (ASCII) par CHDRAW.

Les attributs sont identiques à PLOT en mode caractère.

Application : tracé du carré reliant les sommets (1Ø,1Ø) - (1Ø,2ØØ) - (2ØØ,2ØØ) - (2ØØ.1Ø).

```
                2036 CHDRAW EQU    $2036
                2029 FORME  EQU    $2029
                0010 PLOT   EQU    $10
                000E DRAW   EQU    $0E

6000                        ORG    $6000

6000 7F    2036 INIT    CLR    CHDRAW
6003 86    0F           LDA    #$0F
6005 B7    2029         STA    FORME

6008 CE    6021 DEBUT   LDU    #TABLE
600B AE    C1           LDX    ,U++
600D 10AE  C1           LDY    ,U++
6010 3F                 SWI
6011       10           FCB    PLOT

6012 AE    C1    BOUCLE LDX    ,U++
6014 8C    0004         CMPX   #4
6017 27    07           BEQ    FIN
6019 10AE  C1           LDY    ,U++
601C 3F                 SWI
```

```
601D       0E             FCB     DRAW
601E 20    F2             BRA     BOUCLE

6020 39          FIN      RTS

6021       000A TABLE     FDB     10
6023       000A           FDB     10
6025       000A           FDB     10
6027       00C8           FDB     200
6029       00C8           FDB     200
602B       00C8           FDB     200
602D       00C8           FDB     200
602F       000A           FDB     10

6031       0004           FDB     4

           0000           END

00000 Total Errors



BOUCLE  6012
CHDRAW  2036
DEBUT   6008
DRAW    000E
FIN     6020
FORME   2029
INIT    6000
PLOT    0010
TABLE   6021
```

Après avoir rentré ce programme avec LANGAGE, faites :
EXEC &H6000.

3° Le sous-programme « GETP »

On y accède en tapant &H14 après le code &H3F du SWI.

Les variables d'entrée sont :

– les registres X et Y du 68Ø9.

La variable de sortie est :

– l'accumulateur B du 68Ø9.

Ce programme ramène dans B la couleur du point dont les coordonnées lui ont été passées par les registres X (Ø à 319) et Y (Ø à 199).

Le contenu de B sera donc compris :

– entre – 16 et – 1 s'il s'agit d'une couleur FOND ;

– entre Ø et 15 s'il s'agit d'une couleur FORME.

6. Le sous-programme « GETS »

On y accède en tapant &H1A après le code &H3F du SWI.

Les variables d'entrée sont :

– le registre X et l'accumulateur A du 68Ø9.

La variable de sortie est :

– l'accumulateur B du 68Ø9.

Ce sous-programme renvoie le code ASCII, dans l'accumulateur B, du caractère dont les coordonnées sur l'écran lui ont été passées par X (1 à 4Ø) et A (Ø à 24).

Si le caractère n'appartient pas au code ASCII (par exemple, un GR$), alors B revient avec Ø.

Si le caractère est du type « minuscule accentuée », alors trois appels successifs à GETS sont nécessaires :

– au retour du premier appel, B contiendra &H16 (ACC),

– au retour du second, B contient le code de l'accent, c'est-à-dire :

&H41 accent grave
&H42 accent aigu
&H43 accent circonflexe
&H48 tréma
&H4B cédille

– au retour du troisième, B contient le code ASCII de la lettre minuscule.

7. *Le sous-programme « PUTCH »*

On y accède en tapant &HØ2 après le code &H3F de SWI. La variable d'entrée est l'accumulateur B du 68Ø9. Il y a quatre modes d'appel de PUTCH.

1er mode

● Dans ce mode, le sous-programme PUTCH affiche, à la position courante du curseur, le caractère dont le code ASCII lui est passé par l'accumulateur B, puis avance le curseur d'une case à droite, si le code ASCII est compris entre &H2Ø et &H7F.

● Si le code ASCII est compris entre &HØ7 et &H1F, l'action est la suivante :

&HØ7	BEL	Envoie un BEEP
&HØ8	BS	← Revient une case à gauche
&HØ9	HT	→ Avance d'une case à droite
&HØA	LF	↓ Descend d'une ligne
&HØB	VT	↑ Monte d'une ligne
&HØC	FF	Efface l'écran (RAZ)
&HØD	CR	Retourne à la ligne suivante (ENTREE)
&H11	DC1	Curseur clignotant
&H14	DC4	Arrêt du curseur
&H16	SS2	Procédure d'accentuation
&H18	CAN	Efface la fin de la ligne
&H1B	ESC	Séquence ESCAPE
&H1E	RS	Séparateur d'enregistrement
&H1F	US	Séquence « Unit Separator »

● Si le code ASCII est supérieur à &H7F, le sous-programme PUTCH affiche un GR$. Il faut préalablement mettre dans le pointeur USERAF l'adresse du haut de la table des caractères utilisateurs et ne pas oublier de faire passer à 1 le 3e bit de FLAG de USERAF.

Si B contient &H8Ø, PUTCH affiche le GR$(Ø).

Si B contient &H81, PUTCH affiche le GR$(1)...

2e mode

● Dans ce mode, appelé Séquence US, il faut en premier lieu faire un appel de PUTCH, avec B contenant &H1F (code de US).

On peut ensuite, par deux appels successifs, définir la ligne haute (ou basse) de la fenêtre de travail sur l'écran.

Exemple : pour définir une ligne basse en 21, on réalisera le programme suivant :

```
                0002 PUTCH  EQU    2

6000                        ORG    $6000

6000 CE    600E DEBUT  LDU    #TABLE

6003 E6    C0   BOUCLE LDB    ,U+
6005 C1    04         CMPB   #4
6007 27    04         BEQ    FIN
6009 3F               SWI
600A       02         FCB    PUTCH
600B 20    F6         BRA    BOUCLE

600D 39         FIN    RTS

600E       1F   TABLE  FCB    $1F
600F       20         FCB    $20
6010       24         FCB    $24
6011       04         FCB    4

                0000       END

00000 Total Errors
```

Exemple : pour définir une ligne haute en Ø4, on réalisera le programme suivant :

```
                0002 PUTCH  EQU    2

6000                        ORG    $6000

6000 CE    600E DEBUT  LDU    #TABLE
```

```
6003 E6    C0    BOUCLE LDB     .U+
6005 C1    04           CMPB    #4
6007 27    04           BEQ     FIN
6009 3F                 SWI
600A       02           FCB     PUTCH
600B 20    F6           BRA     BOUCLE

600D 39          FIN    RTS

600E       1F    TABLE  FCB     $1F
600F       12           FCB     $12
6010       11           FCB     $11
6011       04           FCB     4

           0000         END

00000 Total Errors
```

Pour définir la ligne basse, il faut envoyer après US :

– la valeur décimale de la ligne dans les 4 LSB du code, les 4 MSB étant 1,

– la valeur unitaire, de la même façon.

Pour la ligne haute, les 4 MSB valent 2.

Ce mode permet également de positionner le curseur sur l'écran, ligne Y (Ø à 24), colonne X (1 à 4Ø), en ajoutant à Y et X la valeur &H4Ø, puis en rangeant cette valeur dans B avant l'appel de PUTCH.

Exemple : on place le curseur ligne 2 colonne 6.

```
           0002 PUTCH  EQU     2

6000                   ORG     $6000

6000 CE    600E DEBUT  LDU     #TABLE

6003 E6    C0   BOUCLE LDB     .U+
```

```
6005 C1    04            CMPB    #4
6007 27    04            BEQ     FIN
6009 3F                  SWI
600A       02            FCB     PUTCH
600B 20    F6            BRA     BOUCLE

600D 39          FIN     RTS

600E       1F    TABLE   FCB     $1F
600F       42            FCB     $42
6010       46            FCB     $46
6011       04            FCB     4

           0000          END

00000 Total Errors
```

3e mode

Dans ce mode, PUTCH permet l'accentuation des lettres minuscules. Trois appels sont nécessaires pour passer :

- le code SS2 d'accentuation &H16,
- le code de l'accent (voir précédemment)
- le code de la lettre.

Exemple : écriture d'un é (accent algu).

```
           0002 PUTCH   EQU     2

6000                    ORG     $6000

6000 CE    600E DEBUT   LDU     #TABLE

6003 E6    C0   BOUCLE  LDB     ,U+
6005 C1    04              CMPB    #4
6007 27    04              BEQ     FIN
6009 3F                    SWI
600A       02              FCB     PUTCH
```

```
600B 20   F6          BRA    BOUCLE

600D 39        FIN    RTS

600E       16  TABLE  FCB    $16
600F       41         FCB    $41
6010       65         FCB    $65
6011       04         FCB    4

           0000       END

00000 Total Errors



BOUCLE  6003
DEBUT   6000
FIN     600D
PUTCH   0002
TABLE   600E
```

4e mode

Ce mode permet de fixer toutes les caractéristiques de l'écran :

- couleurs du TOUR, du FOND et de la FORME,
- dimension des caractères,
- incrustation, vidéo inverse,
- mode de scrolling.

Chaque appel doit commencer par ESC (&H1B) dans B.

● *Définition des couleurs*

FORME → &H4Ø
FOND → &H5Ø
TOUR → &H6Ø

Les couleurs sont codées de Ø à F

Exemple : une FORME BLEUE → &H44
un FOND ORANGE → &H5F
un TOUR ROSE → &H69...

Le programme suivant définit des caractères BLEUS sur FOND ORANGE et TOUR ROSE :

```
           0002 PUTCH  EQU    2

6000                   ORG    $6000

6000 CE    600E DEBUT  LDU    #TABLE

6003 E6    C0   BOUCLE LDB    ,U+
6005 C1    04          CMPB   #4
6007 27    04          BEQ    FIN
6009 3F                SWI
600A       02          FCB    PUTCH
600B 20    F6          BRA    BOUCLE

600D 39          FIN   RTS

600E       1B    TABLE FCB    $1B
600F       44          FCB    $44
6010       1B          FCB    $1B
6011       5F          FCB    $5F
6012       1B          FCB    $1B
6013       69          FCB    $69
6014       04          FCB    4

           0000        END

00000 Total Errors



BOUCLE  6003
DEBUT   6000
FIN     600D
PUTCH   0002
TABLE   600E
```

● *Définition des attributs spéciaux*

Si à la suite du code ESC (&H1B), on envoie un code compris entre &H7Ø et &H7B, on obtient le résultat ci-dessous :

&H7Ø → caractères de taille normale
&H71 → caractères double hauteur
&H72 → caractères double largeur
&H73 → caractères double dimension
&H74 → mode caractère et couleur
&H75 → mode caractère seulement
&H76 → mode non incrusté
&H77 → mode incrusté
&H78 → scrolling normal
&H79 → scrolling lent
&H7A → mode page
&H7B → vidéo inverse

8. Les sous-programmes de gestion du clavier

1° Lecture rapide : « KTST »

On y accède en tapant &HØC après le code &H3F du SWI.

Les variables de sortie sont :

– le registre code condition CCR du 68Ø9
– l'accumulateur B du 68Ø9.

Ce sous-programme fait une scrutation rapide du clavier et met à Ø le bit Z du CCR si une touche est enfoncée ; sinon, ce bit est à 1.

L'accumulateur B revient avec le code ASCII de la touche enfoncée.

2° Décodage du clavier « GETC »

On y accède en tapant &HØA après le code &H3F du SWI.

Les variables de sortie sont également CCR et B.

Ce sous-programme décode le clavier et commande la fonction d'autorépétition. La latence du clavier étant programmée dans le registre LATCLV (&H2Ø76), on peut inhiber la fonction d'autorépétition en chargeant LATCLV avec une valeur suffisamment grande (par exemple &HFF).

Si la touche (SHIFT) est enfoncée, on accède alors à la partie supérieure des touches.

La touche [CNT] met à Ø le bit 6 du code ASCII de toute autre touche enfoncée (voir les premiers chapitres).

La touche [BASIC] renvoie directement un code numérique par mise à 1 du bit 7 (compris entre &H82 et &HB9).

L'accumulateur B revient donc, soit avec un code ASCII, soit avec le code numérique Basic.

Dans le premier cas, la recherche du caractère se fait dans la table ASCII dont l'adresse est contenue dans le pointeur CHR PTR (&H2Ø6D, 2Ø6E, 2Ø6F). On peut évidemment changer cette adresse (et le flag), ce qui permet de modifier le rôle des touches.

Pour détecter une lettre accentuée, il faut appeler trois fois la routine GETC.

A chaque fois qu'une touche est enfoncée, le buzzer reçoit un BEEP. Ce dernier peut être inhibé par mise à 1 du bit 3 du registre STATUS (&H2Ø19).

9. Le sous-programme « JOYST »

On y accède en tapant &H1C après le code &H3F du SWI.

La variable d'entrée est :

– l'accumulateur A du 68Ø9.

Les variables de sortie sont :

– l'accumulateur B et le CCR du 68Ø9.

Ce sous-programme permet de lire l'état de la manette dont le numéro (Ø ou 1) lui est passé par l'accumulateur A.

B revient avec un nombre compris entre Ø et 8 qui indique l'état du levier, conformément à la figure XI-3.

Le bit de retenue C du CCR est à 1 si le bouton est enfoncé, sinon il est à zéro.

10. Le sous-programme « PRCO »

On y accède en tapant &H24 après le code &H3F du SWI.

Les variables d'entrée sont :

- l'accumulateur B du 68Ø9
- les registres PR.OPC (&H2Ø42), GR CODE (&H2Ø77)

de la page Ø du moniteur.

Les variables de sortie sont :

- le registre CCR du 68Ø9
- le registre PR.STA (&H2Ø43)

de la page Ø du moniteur.

Ce sous-programme gère le module d'interface parallèle (c'est-à-dire surtout l'imprimante).

Le registre PR.OPC contient un octet de code qui conduit aux opérations suivantes :

PR.OPC

%ØØØØØØØ1 → écriture d'un octet
%ØØØØØØ1Ø → copie graphique de l'écran
%ØØØØØ1ØØ → ouverture en mode parallèle
%ØØØ1ØØØØ → fermeture

L'accumulateur B doit contenir le code de l'octet que l'on veut envoyer vers l'imprimante (en général le code ASCII du caractère).

Le registre PR.STA revient avec le code de l'opération réalisée, ou le code d'erreur.

%ØØØØ1ØØØ → dispositif non prêt.

En cas d'erreur, le bit de retenue C du CCR est mis à 1.

Pour le cas où on utilise une imprimante thermique, il faut mettre le code de copie graphique de l'écran dans le registre GRCODE.

Exemple : impression du prénom « Paul » sur l'imprimante parallèle.

```
                0024 PRCO    EQU     $24
                2042 PROPC   EQU     $2042

6000                         ORG     $6000
```

```
6000 CE   6023 DEBUT  LDU    #TABLE
6003 C6   04          LDB    #4
6005 F7   2042        STB    PROPC
6008 3F               SWI
6009      24          FCB    PRCO
600A C6   01          LDB    #1
600C F7   2042        STB    PROPC
600F 3F               SWI
6010      24          FCB    PRCO

6011 E6   C0   BOUCLE LDB    ,U+
6013 C1   04          CMPB   #4
6015 27   04          BEQ    FIN
6017 3F               SWI
6018      24          FCB    PRCO
6019 20   F6          BRA    BOUCLE

601B C6   10   FIN    LDB    #$10
601D F7   2042        STB    PROPC
6020 3F               SWI
6021      24          FCB    PRCO
6022 39               RTS

6023      0F   TABLE  FCB    $0F
6024      50          FCB    $50
6025      61          FCB    $61
6026      75          FCB    $75
6027      6C          FCB    $6C
6028      0D          FCB    $0D
6029      04          FCB    4

          0000        END

00000 Total Errors


BOUCLE  6011
DEBUT   6000
```

```
FIN     601B
PRCO    0024
PROPC   2042
TABLE   6023
```

11. Le sous-programme « NOTE »

On y accède en tapant &H1E après le code &H3F du SWI.

Les variables d'entrée sont :

– l'accumulateur B du 68Ø9

– les registres

OCTAVE (&H2Ø3E, &H2Ø3F)
DUREE (&H2Ø3B, &H2Ø3C)
TEMPO (&H2Ø39, &H2Ø3A)
TIMBRE (&H2Ø3D)

de la page Ø du monlteur.

Ce sous-programme permet de jouer une note dont on définit, dans chaque registre moniteur correspondant, l'octave, la durée, le tempo et le timbre.

OCTAVE (1,16) : nombre de fois que la demi-période de la note fondamentale contenue dans B doit être répétée. Ce registre définit l'octave de la note, et son contenu doit être doublé à chaque fois qu'on descend d'une octave, c'est-à-dire : 1, 2, 4, 8, 16.

DUREE (1,96) : nombre de fois que le diviseur de temps (255) doit être répété, avant de modifier le rapport cyclique (attaque).

96 → ronde
48 → blanche
24 → noire
12 → croche
Ø6 → double croche
Ø3 → triolet

TEMPO (1,255) : définit le tempo de l'ensemble. Donc :

DUREE REELLE TOTALE = TEMPO * DUREE

TIMBRE (Ø,255) : fixe le timbre, de Ø (son continu) à 255 (son le plus prononcé).

Chaque note à transmettre doit être dans l'accumulateur B avant l'appel de NOTE.

Les codes des notes sont les suivants :

PAUSE → &H30	MI → &H35	LA → &H3A
DO → &H31	FA → &H36	LA# → &H3B
DO# → &H32	FA# → &H37	SI → &H3C
RE → &H33	SO → &H38	
RE# → &H34	SO# → &H39	

12. *Les sous-programmes « LPIN » et « GETL »*

LPIN

On y accède en tapant &H16 après le code &H3F du SWI

Ce sous-programme teste l'état du bouton-interrupteur du crayon optique et met à 1 le bit de retenue C du CCR s'il est enfoncé. Sinon, C est à Ø.

GETL

On y accède en tapant &H18 après le code &H3F du SWI.

Les variables de sortie sont :

- les registres X et Y du 6809
- le registre CCR du 6809.

Ce sous-programme lit les coordonnées du point de l'écran situé face au phototransistor du crayon optique et les range dans X (Ø à 319) et Y (Ø à 199).

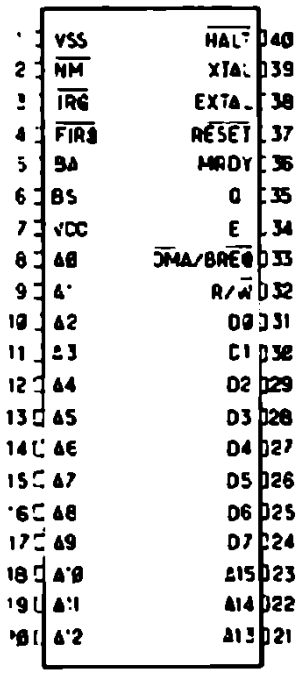
Si la mesure est correcte, le bit de retenue est mis à Ø. Sinon, il est mis à 1.

Annexes

Annexe 1

LE 6809

ANNEXE 1 LE 6809



Brochage

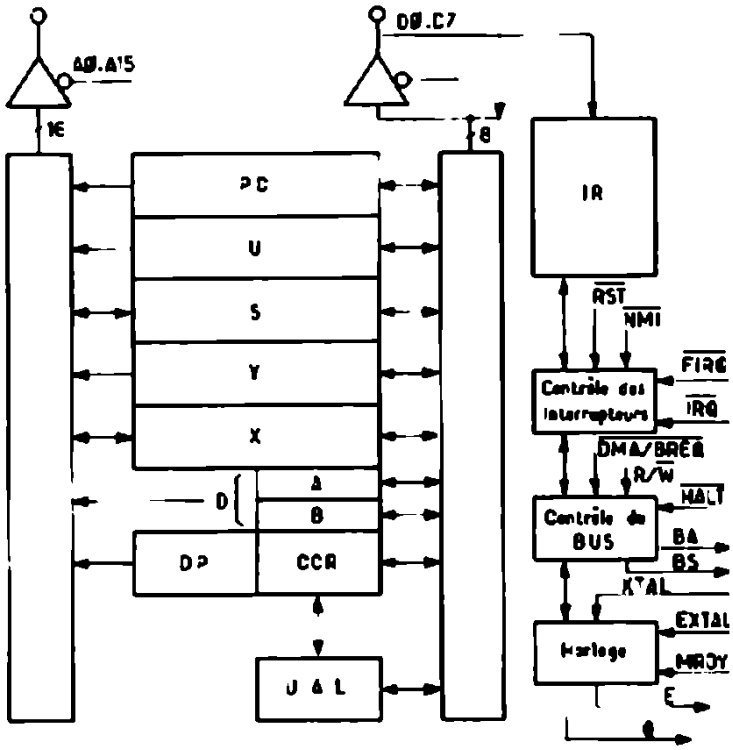


Schéma fonctionnel

E	F	H	I	N	Z	V	C

Registre Codes Condition

BA	BS	
Ø	Ø	Normal
Ø	1	Reconnaissance d'interruption
1	Ø	Reconnaissance de SYNC
1	1	Arrêt

Etat du 6809

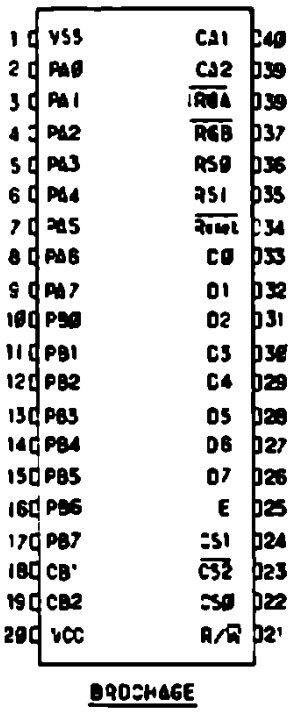
FFFE	FFFF	$\overline{\text{RESET}}$
FFFC	FFFD	$\overline{\text{NMI}}$
FFFA	FFFB	SWI
FFF8	FFF9	$\overline{\text{IRQ}}$
FFF6	FFF7	$\overline{\text{FIRQ}}$
FFF4	FFF5	SWI2
FFF2	FFF3	SWI3
FFF0	FFF1	Réservé

Pointeurs d'interruption

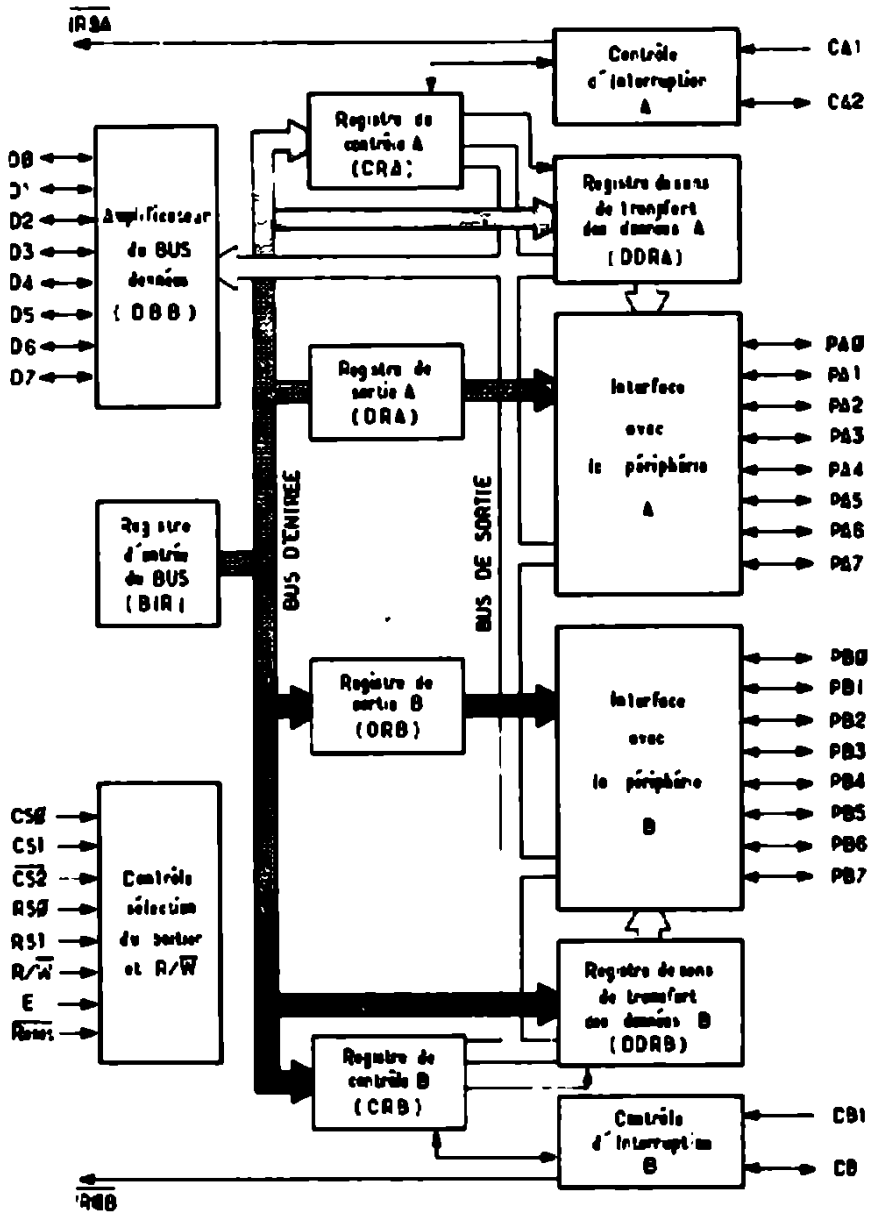
Note : Les demandes NMI, FIRQ et IRQ sont mémorisées sur le front descendant de Q, sauf pendant les fonctionnements en vol de cycle (exemple DMA) au cours desquels seule NMI est mémorisée.

Annexe 2

LE 6821



BROCHAGE



SCHEMA FONCTIONNEL

RS1	RSØ	CRA-2	CRB-2	Registre sélecté
Ø	Ø	1	x	ORA
Ø	Ø	Ø	x	DDRA
Ø	1	x	x	CRA
1	Ø	x	1	ORB
1	Ø	x	Ø	DDRB
Ø	Ø	x	x	CRB

Adressage interne

7	6	5 4 3	2	1 Ø	
IRQA1	IRQA2	Contrôle de CA2	Accès à DDRA	Contrôle de CA1	CRA

7	6	5 4 3	2	1 Ø	
IRQB1	IRQB2	Contrôle de CB2	Accès à DDRB	Contrôle de CB1	CRB

Format des mots de contrôle

Annexe 3

LA CARTE MEMOIRE DU MO5

ADRESSE (hexa)			
0000	–	1FFF	Mémoire d'écran 8 K
2000	–	20FF	Page zéro du moniteur
2100	–	21FF	Page zéro du Basic
2200	–	5FFF	Mémoire utilisateur
6000	–	9FFF	Idem
A000	–	A7BF	1,9 Ko pour le floppy
A7C0	–	A7C3	PIA 6821 système
A7C4	–	A7CB	LIBRE...
A7CC	–	A7CF	PIA 6821 extension jeux
A7D0	–	A7DF	Contrôleur mini-floppy
A7E0	–	A7E3	PIA 6821 Interface parallèle
A7E4	–	A7E7	Gate Array
A7E8	–	A7FF	Libre pour PIA d'extensions
A800	–	AFFF	2 Ko libres...
B000	–	EFFF	Cartouche de ROM d'applications
C000	–	EFFF	Basic (libre de $B000 à $BFFF)
F000	–	FFFF	Moniteur 4 Ko

Annexe 4

PAGE Ø DU MONITEUR

Adresse	Identificateur	Commentaire
$2019	STATUS	b4 = point graphique b3 = masque du buzzer
$201B	RANG	Numéro de ligne du curseur
$201C	COLN	Numéro de colonne du curseur
$201E	TOPRAN	Premier numéro de ligne logique de la fenêtre
$2020	BOTRAN	Dernier numéro de ligne logique de la fenêtre
$2029	FORME	Registre couleur (– 16 à + 15) pour PLOT et DRAW
$202B	COLOUR	Registre couleur [PBVRPBVR] P = 1 → PASTEL
$2036	CHDRAW	Code ASCII du caractère pour PLOT et DRAW
$2039	TEMPO,2	Tempo pour NOTE
$203B	DUREE,2	Durée pour NOTE
$203D	TIMBRE	Timbre pour NOTE
$203E	OCTAVE,2	Octave pour NOTE
$2042	PROPC	Commande pour imprimante
$2043	PRSTA	Etat de l'imprimante
$2048	DKOPC	Commande pour le floppy
$2049	DKDRV	Numéro du drive
$204A	DKTRK,2	Numéro de piste
$204C	DKSEC	Numéro du secteur
$204E	DKSTA	Etat courant du contrôleur
$204F	DKBUF,2	Pointeur du buffer d'E/S
$205E	SWI1PT,2	Pointeur de SWI1
$2061	TIMEPT,3	Pointeur et flag de la tempo d'interruption utilisateur
$2064	IRQPT,3	Pointeur et flag de IRQ
$2067	FIRQPT,3	Pointeur et flag de FIRQ
$206A	SIMUL,3	Pointeur et flag de la table d'entrée dans le moniteur
$206D	CHRPTR,3	Pointeur et flag de la table de décodage du clavier
$2070	USERAF,3	Pointeur et flag de la table des caractères GR$
$2073	GENPTR,3	Pointeur et flag du générateur de caractères
$2076	LATCLV	Latence du clavier
$2077	GRCODE	Mot de code pour la copie graphique de l'écran

Annexe 5

LES ADRESSES IMPORTANTES

Adresse	Identificateur	Commentaire
		Adresses de l'écran
$0000	STAD	Premier octet
$1F40	ENDAD	Dernier octet
		6821 système
$A7C0	PRA	Registre PORT A
$A7C1	PRB	Registre PORT B
$A7C2	CRA	Registre de contrôle du PORT A
$A7C3	CRB	Registre de contrôle du PORT B
		6821 jeux
$A7CC	PRA1	Registre PORT A PA0-7 (entrée) lecture des manettes
$A7CD	PRB1	Registre PORT B PB0-5 (sortie) convertisseur N/A PB6 bouton manette 0 PB7 bouton manette 1
$A7CE	CRA1	Registre de contrôle du PORT A1
$A7CF	CRB1	Registre de contrôle du PORT B1

Codes d'appel des S.P. Moniteur

MENU	.0	PLOT	.$10		
PUTC	.2	CHPL	.$12	GETS	.$1A
GETC	.$A	GETP	.$14	JOYS	.$1C
KTST	.$C	LPIN	.$16	NOTE	.$1E
DRAW	.$E	GETL	.$18	PRCO	.$24

Annexe 6

LES PRINCIPAUX PARAMETRES

I – Paramètres musicaux

P	$30	RE#	$34	SO	$38	SI	$3C
DO	$31	MI	$35	SO#	$39		
DO#	$32	FA	$36	LA	$3A		
RE	$33	FA#	$37	LA#	$3B		

II – Paramètres imprimante

00000001	Ecriture d'un octet
00000010	Copie graphique d'écran
00000100	Ouverture en parallèle
00010000	Fermeture
00001000	Défaut (PRSTA)

Annexe 7

JEU DE CODES ASCII

Code décimal	Code hexa	
00	00	NUL
01	01	SON
02	02	START OF TEXT
03	03	STOP (CONT/C)
04	04	END OF TRANS.
05	05	ENQUIRE
06	06	ACKNOWLEDGE
07	07	SONNETTE
08	08	BS (←)
09	09	HT (→)
10	0A	LF (↓)
11	0B	VT (↑)
12	0C	FF (RAZ)
13	0D	CR (ENTREE)
14	0E	SO (semi-graph.)
15	0F	SI (alphanum.)
16	10	DLE
17	11	DC1 (clignot. curs.)
18	12	DC2 (répétition)
19	13	DC3
20	14	DC4 (arrêt curs.)
21	15	NAK
22	16	SS2 (accent.)
23	17	ETB
24	18	CAN (CONT/X)
25	19	EM
26	1A	SUB
27	1B	ESC
28	1C	INS (INS)
29	1D	DEL (EFF)
30	1E	RS ( )
31	1F	US
32	20	Espace
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%

Code décimal	Code hexa	
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K

Code décimal	Code hexa	
76	4C	L
77	4D	M
78	4E	N
79	4F	O
80	50	P
81	51	Q
82	52	R
83	53	S
84	54	T
85	55	U
86	56	V
87	57	W
88	58	X
89	59	Y
90	5A	Z
91	5B	[
92	5C	
93	5D	]

Code décimal	Code hexa	
94	5E	↑
95	5F	←
96	60	
97	61	a
98	62	b
99	63	c
100	64	d
101	65	e
102	66	f
103	67	g
104	68	h
105	69	i
106	6A	j
107	6B	k
108	6C	l
109	6D	m
110	6E	n

Code décimal	Code hexa	
111	6F	o
112	70	p
113	71	q
114	72	r
115	73	s
116	74	t
117	75	u
118	76	v
119	77	w
120	78	x
121	79	y
122	7A	z
123	7B	
124	7C	:
125	7D	|
126	7E	≈
127	7F	DEL

Annexe 8

CODES CLAVIER DU BASIC

$82	→ Stop	$95	→ /	$A7	→ L
$83	→ SS2	$96	→ S	$A8	→ RS
$85	→ RC	$97	→ B	$A9	→ .
$86	→ FF	$98	→ LF	$AA	→ 5
$87	→ C	$99	→ Espace	$AB	→ 8
$88	→ VT	$9A	→ 3	$AC	→ T
$89	→ W	$9B	→ Ø	$AD	→ I
$8A	→ 1	$9C	→ E	$AE	→ G
$8B	→ +	$9D	→ P	$AF	→ K
$8C	→ A	$9E	→ D	$BØ	→ Insert
$8D	→ *	$9F	→ M	$B1	→ ,
$8E	→ Q	$AØ	→ HT	$B2	→ 6
$8F	→ V	$A1	→ @	$B3	→ 7
$9Ø	→ BS	$A2	→ 4	$B4	→ Y
$91	→ X	$A3	→ 9	$B5	→ U
$92	→ 2	$A4	→ R	$B6	→ H
$93	→ –	$5	→ O	$B7	→ J
$94	→ Z	$A6	→ F	$B8	→ Delete
				$B9	→ N

Annexe 9

LES CODES D'ERREURS

1	NEXT sans FOR
2	Erreur de syntaxe
3	RETURN sans GOSUB
4	READ sans DATA
5	Argument incorrect
6	Nombre trop grand
7	Dépassement mémoire
8	Numéro de ligne inexistant
9	Indice trop grand
10	Redimensionnement d'un tableau
11	Division par
12	Instruction interdite en mode direct
13	Nom de variable faux
14	Manque de place pour les chaînes
15	Chaîne trop complexe
16	Formule trop complexe
17	Suite de l'exécution impossible
18	Fonction non définie
19	RESUME absent
20	RESUME sans ERROR
21	Erreur indéfinie
22	Opérande absent
23	FOR sans NEXT
50	Numéro de fichier incorrect
51	Mode d'accès incorrect
52	Fichier déjà ouvert
53	Erreur d'E/S
54	Lecture après fermeture
55	Descripteur de fichier incorrect
56	Commande dans un fichier en chargement
57	Fichier non ouvert
58	Données fausses
59	Périphérique occupé
60	Périphérique absent
61	Programme protégé

Annexe 10

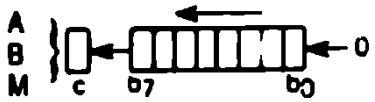
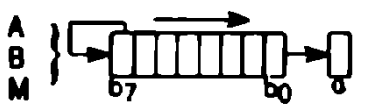
LE JEU D'INSTRUCTIONS DU 6809

Instructions de branchement

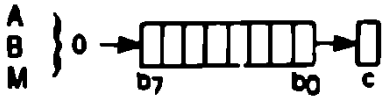
Instruction	Forms	Addressing Mode			Description	5	3	2	1	0
		Relative								
		Op	~	≠		H	N	Z	V	C
BCC	BCC LBCC	24 10 24	3 5(6)	2 4	Branch C = 0 Long Branch C = 0	● ●	● ●	● ●	● ●	● ●
BCS	BCS LBCS	25 10 25	3 5(6)	2 4	Branch C = 1 Long Branch C = 1	● ●	● ●	● ●	● ●	● ●
BEQ	BEQ LBEQ	27 10 27	3 5(6)	2 4	Branch Z = 1 Long Branch Z = 1	● ●	● ●	● ●	● ●	● ●
BGE	BGE LBGE	2C 10 2C	3 5(6)	2 4	Branch ≥ Zero Long Branch ≥ Zero	● ●	● ●	● ●	● ●	● ●
BGT	BGT LBGT	2E 10 2E	3 5(6)	2 4	Branch > Zero Long Branch > Zero	● ●	● ●	● ●	● ●	● ●
BHI	BHI LBHI	22 10 22	3 5(6)	2 4	Branch Higher Long Branch Higher	● ●	● ●	● ●	● ●	● ●
BHS	BHS LBHS	24 10 24	3 5(6)	2 4	Branch Higher or Same Long Branch Higher or Same	● ●	● ●	● ●	● ●	● ●

Instruction	Forms	Addressing Mode			Description	5	3	2	1	0
		Relative								
		Op	~	#		H	N	Z	V	C
BLE	BLE LBLE	2F 10 2F	3 5(6)	2 4	Branch ≤ Zero Long Branch ≤ Zero	● ●	● ●	● ●	● ●	● ●
BLO	BLO LBLO	25 10 25	3 5(6)	2 4	Branch Lower Long Branch Lower	● ●	● ●	● ●	● ●	● ●
BLS	BLS LBLS	23 10 23	3 5(6)	2 4	Branch Lower or Same Long Branch Lower or Same	● ●	● ●	● ●	● ●	● ●
BLT	BLT LBLT	2D 10 2D	3 5(6)	2 4	Branch < Zero Long Branch < Zero	● ●	● ●	● ●	● ●	● ●
BMI	BMI LBMI	2B 10 2B	3 5(6)	2 4	Branch Minus Long Branch Minus	● ●	● ●	● ●	● ●	● ●
BNE	BNE LBNE	26 10 26	3 5(6)	2 4	Branch Z = 0 Long Branch Z = 0	● ●	● ●	● ●	● ●	● ●
BPL	BPL LBPL	2A 10 2A	3 5(6)	3 4	Branch Plus Long Branch Plus	● ●	● ●	● ●	● ●	● ●
BRA	BRA LBRA	20 16	3 5	2 3	Branch Always Long Branch Always	● ●	● ●	● ●	● ●	● ●
BRN	BRN LBRN	21 10 21	3 5	2 4	Branch Never Long Branch Never	● ●	● ●	● ●	● ●	● ●
BSR	BSR LBSR	8D 17	7 9	2 3	Branch to Subroutine Long Branch to Subroutine	● ●	● ●	● ●	● ●	● ●
BVC	BVC LBVC	28 10 28	3 5(8)	2 4	Branch V = 0 Long Branch V = 0	● ●	● ●	● ●	● ●	● ●
BVS	BVS LBVS	29 10 29	3 5(8)	2 4	Branch V = 1 Long Branch V = 1	● ●	● ●	● ●	● ●	● ●

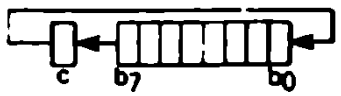
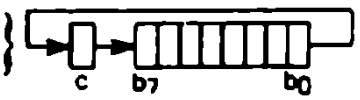
Instruction	Forms	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
ABX													
ADC	ADCA ADCB	89 C9	2 2	2 2	99 D9	4 4	2 2	A9 E9	4+ 4+	2+ 2+	B9 F9	5 5	3 3
ADD	ADDA ADDB ADDD	8B CB C3	2 2 4	2 2 3	9B DB D3	4 4 6	2 2 2	AB EB E3	4+ 4+ 6+	2+ 2+ 2+	BB FB F3	5 5 7	3 3 3
AND	ANDA ANDB ANDCC	84 C4 1C	2 2 3	2 2 2	94 D4	4 4	2 2	A4 E4	4+ 4+	2+ 2+	B4 F4	5 5	3 3
ASL	ASLA ASLB ASL				08	6	2	68	6+	2+	78	7	3
ASR	ASRA ASRB ASR				07	6	2	67	6+	2+	77	7	3
BIT	BITA BITB	65 C5	2 2	2 2	95 D5	4 4	2 2	A5 E5	4+ 4+	2+ 2+	B5 F5	5 5	3 3
CLR	CLRA CLRB CLR				0F	6	2	6F	6+	2+	7F	7	3
CMP	CMPA CMPB CMPD CMPS CMPU CMPX CMPY	81 C1 10 83 11 8C 11 83 8C 10 8C	2 2 5 5 5 4 5	2 2 4 4 4 3 4	91 D1 10 93 11 9C 11 93 9C 10 9C	4 4 7 7 7 6 7	2 2 3 3 3 2 3	A1 E1 10 A3 11 AC 11 A3 AC 10 AC	4+ 4+ 7+ 7+ 7+ 6+ 7+	2+ 2+ 3+ 3+ 3+ 2+ 3+	B1 F1 10 B3 11 BC 11 B3 BC 10 BC	5 5 8 6 6 7 6	3 3 4 4 4 3 4
COM	COMA COMB COM				03	6	2	63	6+	2+	73	7	3
CWAI		3C	≥20	2									
DAA													

Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
3A	3	1	B + X → X (Unsigned)	•	•	•	•	•
			A + M + C → A B + M + C → B	↕ ↕	↕ ↕	↕ ↕	↕ ↕	↕ ↕
			A + M → A B + M → B D + M:M + 1 → D	↕ ↕ •	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕
			A . M → A B . M → B CC . IMM → CC	• •	↕ ↕	↕ ↕	0 0	• • 7
48 58	2 2	↑ ↑	A B M 	8 8 8	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕
47 57	2 2	↑ ↑	A B M 	8 8 8	↕ ↕ ↕	↕ ↕ ↕	• • •	↕ ↕ ↕
			Bit Test A (M – A) Bit Test B (M – B)	• •	↕ ↕	↕ ↕	0 0	• •
4F 5F	2 2	↑ ↑	0 → A 0 → B 0 → M	• • •	0 0 0	1 1 1	0 0 0	0 0 0
			Compare M from A Compare M from B Compare M:M + 1 from D Compare M:M + 1 from S Compare M:M + 1 from U Compare M:M + 1 from X Compare M:M + 1 from Y	8 8 • • • • •	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ • • •
43 53	2 2	1 1	$\overline{A}$ → A $\overline{B}$ → B $\overline{M}$ → M	• • •	• • •	↕ ↕ ↕	0 0 0	1 1 1
			CC / IMM → CC Wait for Interrupt					7
19	2	1	Decimal Adjust A	•	•	↕	0	1

Instruction	Forms	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
DEC	DECA DECB DEC				0A	6	2	6A	6+	2+	7A	7	3
EOR	EORA EORB	88 C8	2 2	2 2	98 D8	4 4	2 2	A8 E8	4+ 4+	2+ 2+	B8 F8	5 5	3 3
EXG	R1, R2	1E	8	2									
INC	INCA INCB INC				0C	6	2	6C	6+	2+	7C	7	3
JMP					0E	3	2	6E	3+	2+	7E	4	3
JSR					9D	7	2	AD	7+	2+	BD	8	3
LD	LDA LDB LDD LDS LDU LDX LDY	86 C6 CC 10 CE CE 8E 10 8E	2 2 3 4 3 3 4	2 2 3 4 3 3 4	96 D6 DC 10 DE DE 9E 10 9E	4 4 5 6 5 5 6	2 2 2 3 2 2 3	A6 E6 EC 10 EE EE AE 10 AE	4+ 4+ 5+ 6+ 5+ 5+ 6+	2+ 2+ 2+ 3+ 2+ 2+ 3+	B6 F6 FC 10 FE FE BE 10 BE	5 5 6 7 6 6 7	3 3 3 4 3 3 4
LEA	LEAS LEAU LEAX LEAY							32 33 30 31	4+ 4+ 4+ 4+	2+ 2+ 2+ 2+			
LSL	LSLA LSLB LSL				08	6	2	68	6+	2+	78	7	3
LSR	LSRA LSRB LSR				04	6	2	64	6+	2+	74		3
MUL													
NEG	NEGA NEGB NEG				00	6	2	60	6+	2+	70	7	3

Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
4A 5A	2 2	1 1	A – 1 → A B – 1 → B M – 1 → M	• • •	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	• • •
			A ∀ M → A B ∀ M → A	• •	↕ ↕	↕ ↕	0 0	• •
			R1 → R2(2)	•	•	•	•	•
4C 5C	2 2	1 1	A + 1 → A B + 1 → B M + 1 → M	• • •	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	• • •
			EA(3) → PC	•	•	•	•	•
			Jump to Subroutine	•	•	•	•	•
			M → A M → B M:M + 1 → D M:M + 1 → S M:M + 1 → U M:M + 1 → X M:M + 1 → Y	• • • • • • •	↕ ↕ ↕ ↕ ↕ ↕ ↕	↕ ↕ ↕ ↕ ↕ ↕ ↕	0 0 0 0 0 0 0	• • • • • • •
			EA(3) → S EA(3) → U EA(3) → X EA(3) → Y	• • • •	• • • •	• • ↕ ↕	• • • •	• • • •
48 58	2 2	1 1	A B M 	• • •	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕
44 54	2 2	1 1	A B M 	• • •	0 0 0	↕ ↕ ↕	• • •	↕ ↕ ↕
3D	11	1	A x B → D (Unsigned)	•	•	↕	•	9
40 50	2 2	1 1	$\overline{A}$ + 1 → A $\overline{B}$ + 1 → B $\overline{M}$ + 1 → M	8 8 8	↕ ↕ ↕	↕ 1 ↕	↕ ↕ ↕	↕ ↕ ↕

Instruction	Forms	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
NOP													
OR	ORA ORB ORCC	8A CA 1A	2 2 3	2 2 2	9A DA	4 4	2 2	AA EA	4+ +	2+ 2+	BA FA	5 5	3 3
PSH	PSHS PSHU	34 36	5+4 5+4	2 2									
PUL	PULS PULU	35 37	5+4 5+4	2 2									
ROL	ROLA ROLB ROL				09	6	2	69	6+	2+	79	7	3
ROR	RORA RORB ROR				06	6	2	66	6+	2+	76	7	3
RTI													
RTS													
SBC	SBCA SBCB	82 C2	2 2	2 2	92 D2	4 4	2 2	A2 E2	4+ 4+	2+ 2+	B2 F2	5 5	3 3
SEX													
ST	STA STB STD STS STU STX STY				97 D7 DD 10 DF DF 9F 10 9F	4 4 5 6 5 5 6	2 2 2 3 2 2 3	A7 E7 ED 10 EF EF AF 10 AF	4+ 4+ 5+ 6+ 5+ 5+ 6+	2+ 2+ 2+ 3+ 2+ 2+ 3+	B7 F7 FD 10 FF FF BF 10 BF	5 5 6 7 6 6 7	3 3 3 4 3 3 4
SUB	SUBA SUBB SUBD	80 C0 83	2 2 4	2 2 3	90 D0 93	4 4 6	2 2 2	A0 E0 A3	4+ 4+ 6+	2+ 2+ 2+	B0 F0 B3	5 5 7	3 3 3

Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
12	2	1	No Opération	●	●	●	●	●
			A V M → A B V M → B CC V IMM → CC	● ●	↕ ↕	↕ ↕	0 0 7	● ●
			Push Registers on S Stack Push Registers on U Stack	● ●	● ●	● ●	● ●	● ●
			Pull Registers from S Stack Pull Registers from U Stack	● ●	● ●	● ●	● ●	● ●
49 59	2 2	1 1	A B M 	● ● ●	↕ ↕ ↕	↕ ↕ ↕	• • •	↕ ↕ ↕
46 56	2 2	1 1	A B M 	● ● ●	↕ ↕ ↕	↕ ↕ ↕	● ● ●	↕ ↕ ↕
3B	6/15	1	Return from Interrupt					7
39	5	1	Return from Subroutine	●	●	●	●	●
			A - M - C → A B - M - C → B	↕ ↕	↕ ↕	↕ ↕	↕ ↕	↕ •
1D	2	1	Sign Extend B into A	●	•	↕	0	●
			A → M B → M D → M:M + 1 S → M:M + 1 U → M:M + 1 X → M:M+1 Y → M:M + 1	● ● ● ● ● ● ●	• • • • • • •	↕ ↕ ↕ ↕ ↕ ↕ ↕	0 0 0 0 0 0 0	● ● ● ● ● ● ●
			A - M → A B - M → B D - M:M + 1 → D	↕ ↕ ●	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕	↕ ↕ ↕

Instruction	Forms	Addressing modes											
		Immediate			Direct			Indexed			Extended		
		Op	~	#	Op	~	#	Op	~	#	Op	~	#
SWI	SWI(6) SWI2(6) SWI3(6)												
SYNC													
TFR	R1, R2	1F	6	2									
TST	TSTA TSTB TST				0D	6	2	6D	6+	2+	7D	7	3

Legend :		M	Complement of M	↑	Test and set if true, cleared otherwise
OP	Opération Code (Hexa)	→	Transfer into	•	Not Affected
~	Number of MPU Cycles	H	Half-carry (from bit 3)	CC	Condition Code Register
#	Number of Program Bytes	N	Negative (sign bit)	:	Concatenation
+	Arithmetic Plus	Z	Zero (Reset)	V	Logical or
-	Arithmetic Minus	V	Overflow, 2's complement	.	Logical and
.	Multiply	C	Carry from ALU	∀	Logical Exclusive or

Inherent			Description	5	3	2	1	0
Op	~	#		H	N	Z	V	C
3F 10 3F 11 3F	19 20 20	1 2 1	Software Interrupt 1 Software Interrupt 2 Software Interrupt 3	● ● ●	● ● ●	● ● ●	● ● ●	● ● ●
13	≥ 4	1	Synchronize to Interrupt	●	●	●	●	●
			R1 → R2 (2)	●	●	●	●	●
4D 5D	2 2	1 1	Test A Test B Test M	● ● ●	* * *	1 1 1	0 0 0	● ● ●

Notes :

1. This column gives a base cycle and byte count. To obtain total count, add the values obtained from the INDEXED ADDRESSING MODE table, in Appendix F.
2. R1 and R2 may be any pair of 8 bit or any pair of 16 bit registers
   The 8 bit registers are : A, B, CC, DP
   The 16 bit registers are : X, Y, U, S, D, PC
3. EA is the effective address.
4. The PSH and PUL instructions require 5 cycles plus 1 cycle for each byte pushed or pulled.
5. 5(6) means : 5 cycles if branch not taken, 6 cycles if taken (Branch Instructions).
6. SWI sets I and F bits. SWI2 and SWI3 do not affect I and F.
7. Conditions Codes set as a direct result of the instruction.
8. Value of half-carry flag is undefined.
9. Special Case – Carry set if b7 is SET.

Index

Mot	Page
A	
ABS	66
AND	59
ASC	76
ATTRB	37
B	
BEEP	88
BOX	109
BOXF	109
C	
CHR$	76
CINT	68
CLEAR	50, 62, 76
CLOSE	90
CLS	37
COLOR	36, 110
CONSOLE	38
CONT	103
COS	68
CSRLIN	44
D	
DATA	30
DEF	49, 66
DELETE	103
DIM	81

Mot	Page
E	
ELSE	58
EOF	90
EQV	59
ERL	101
ERR	101
ERROR	101
EXEC	62
EXP	68
F	
FIX	66
FOR	59
FRE	76
G	
GO	57, 103
GR$	49
I	
IF	58
IMP	59
INKEY$	29
INPEN	85
INPUT	27, 29, 86, 90
INSTR	76
INT	66, 68

Mot	Page
L	
LEFT$	76
LEN	77
LINE	28, 43, 90, 109
LIST	90, 91, 104
LOAD	91, 93
LOCATE	42
LOG	66
M	
MERGE	91
MID$	77
MOD	67
MOTOR	91
N	
NEW	104
NEXT	59
NOT	59
O	
OFF	91
ON	57, 62, 91
OPEN	92
OR	59

Mot	Page
P	
PEEK	100
PEN	85, 86
PLAY	88
POINT	110
POKE	100
POS	43, 92
PRINT	32, 33, 92
PSET	42, 110
PTRIG	86
R	
READ	30
RESTORE	31
RESUME	102
RETURN	62
RIGHT$	77
RND	68
RUN	104

Mot	Page
S	
SAVE	92, 93, 104
SCREEN	42, 44, 110
SGN	67, 68
SIN	68
SKIPF	92
SPC	32
SQR	68
STEP	59
STICK	87
STRIG	87
STR$	77
SUB	61

Mot	Page
T	
TAB	32
TAN	68
THEN	58
TO	59, 57, 103
TROFF	104
TRON	104
TUNE	87
U	
USING	33
V	
VAL	77
VARPTR	100
X	
XOR	59

Collection Micro-Systèmes - ETSF **M. OURY**

MAITRISEZ LES TO 7 ET TO 7-70
du Basic au langage machine

Cet ouvrage s'adresse aussi bien au **débutant,** qui y trouvera une description tres détaillée du Basic des TO 7 et TO 7-70 avec de nombreux programmes d'applications, qu'au **programmeur averti,** qui vise déjà la **programmation en Assembleur** et la fabrication de ses propres extensions. Vous lirez avec intérêt **la présentation complète du 6809** et de son mode d'adressage, vous découvrirez toutes les implantations mémoires, tous les branchements de connecteurs et les adresses des sous-programmes du moniteur.

Principaux chapitres :

- **Materiel, logiciel et extensions.**
- **Bases du langage.**
- **Gestion de l'écran.**
- **Caractères utilisateurs.**
- **Branchements, itérations et sous-programmes.**
- **Variables numériques, chaines et tableaux.**
- **Dessins et graphiques.**
- **Le microprocesseur 6809 : présentation détaillée et programmation en Assembleur.**

Un livre de 200 pages, format 15 × 21 cm

Edité par :

EDITIONS TECHNIQUES ET SCIENTIFIQUES FRANÇAISES
2 a 12, rue de Bellevue · 75940 PARIS CEDEX 19